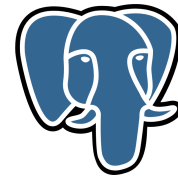


# Documentation

## Serveur de bases de données PostgreSQL pour GDA

**GDA Score**

General  
Data  
Anonymity



Introduction	2
Installation	3
Lier le serveur à GDA Score	3
Le Github gda_insa	3
△ Le fichier master.json △	4
La structure du fichier	4
Un master.json par source de données	6
Créer un master.json pour votre nouvelle source de données	7
Utiliser une source de données	8
Gestion des bases de données	10
Première utilisation	10
Créer une base	10
Créer des tables	11
Populer une base à partir d'un fichier	11
Supprimer des tables / une base	12

# I. Introduction

Le but de l'implémentation d'un serveur de bases de données est de pouvoir utiliser et tester des nouvelles bases dans les différents programmes d'évaluation de techniques d'anonymisation avec le framework GDA. Ainsi, que ce soit pour des attaques, de l'évaluation d'utilité, ou tout autre programme de GDA, il est possible d'utiliser les bases de données sur ce serveur PostgreSQL. Il vient alors en complément du serveur PostgreSQL déjà hébergé par GDA, qui lui possède un set de bases défini, ce qui limite les possibilités de test avec le framework.

Ce nouveau serveur a alors comme objectif d'étoffer le nombre de bases anonymisées, que ce soit dans l'objectif de tester des techniques d'anonymisation connues sur de nouvelles bases à structure ou données différentes, ou dans l'objectif de tester de nouvelles techniques d'anonymisation.

Dans l'optique de rendre son utilisation la plus simple possible, nous avons créé un script, en python, qui permettra de gérer les bases de données sur le serveur. Ce script ne vous sera utile que si vous avez l'intention de modifier le serveur, par l'ajout, la suppression, la population de tables/bases. Par exemple, si vous avez une base anonymisée que vous voulez partager sur le serveur, ce script vous sera utile. À l'inverse, si vous travaillez sur une attaque, et donc seulement de la "lecture" de bases, vous n'aurez pas besoin du script.

## Vocabulaire :

- *Source de données*, ou *datasource* : Correspond au nom qui permettra d'identifier plusieurs bases de données avec la même structure (mêmes tables), mais avec les données sous différentes formes (anonymisées différemment). (ex. la source de données **banking** permet d'identifier toutes les bases de type banking, comme raw\_banking, k\_anon\_5\_banking...)
- *Base de données* : Dans notre cas, cela correspondra à une base de donnée appartenant à une source de données (ex. raw\_banking appartient à la source banking)

## II. Installation

### A. Lier le serveur à GDA Score

Afin de pouvoir utiliser GDA Score avec notre serveur hébergé à l'INSA il est tout d'abord nécessaire d'avoir un dossier de configuration contenant le fichier *master.json* à un emplacement connu par vous et par GDA.

Pour cela il vous suffit d'exécuter la commande *gdascore\_init* afin d'y spécifier le chemin voulu pour l'emplacement du dossier *config*.

Maintenant que vous avez un chemin défini pour la config, donc pour le *master.json*, vous pouvez vous référer à la partie [III.C](#) pour créer une nouvelle source de données, ou [III.D](#) pour utiliser une source de données.

Chaque groupe possède ses propres identifiants que nous avons communiqué au chef de groupe. Il faudra modifier les variables d'environnement `GDA_SCORE_RAW_USER` et `GDA_SCORE_RAW_PASS` avec vos propres identifiants.

△ Si vous souhaitez réutiliser le serveur psql de GDA, il vous faudra modifier identifiant et mot de passe dans les variables d'environnement pour remettre ceux de GDA.

L'adresse du serveur est : **sds5000.insa-cvl.fr**

Le port est : **3700**

### B. Le Github *gda\_insa*

Pour permettre le partage et stocker le script, un Github est disponible à l'adresse suivante : [https://github.com/Alex2mars/gda\\_insa](https://github.com/Alex2mars/gda_insa).

Ce Github aura l'architecture suivante :

- Dossier "scripts" : Dossier qui contiendra le script de gestion
- Dossier "masters json" : Dossier avec les différents *master.json* pour les nouvelles sources de données (voir chapitre sur le *master.json*)
- *master.json.template* : Fichier modèle utile lors de la création d'une nouvelle source de données

Nous vous conseillons de le cloner en local afin de récupérer tous les éléments simplement.

### III. ▲ Le fichier *master.json* ▲

Pour pouvoir utiliser une nouvelle source de données avec GDA, elle doit être renseignée dans le fichier *master.json*. Ce fichier permet à GDA de répertorier les bases existantes, de les assigner à des techniques d'anonymisation pour ensuite pouvoir les utiliser dans un programme.

Comme ce nouveau serveur ne contiendra que des bases inconnues pour GDA, de source de données nouvelles, à chaque fois que l'on voudra utiliser une de ces nouvelles bases, il faudra adapter le *master.json* en fonction.

#### A. La structure du fichier

Il est important de savoir comment est structuré ce fichier, afin de pouvoir le modifier ou l'utiliser au mieux. Le fichier *master.json* peut être divisé en 3 parties : (vous pouvez ouvrir votre *master.json* local en parallèle pour le comprendre)

- “services” : Cette partie est utile pour renseigner les différents services pouvant héberger des données. Il sert notamment à renseigner le serveur PostgreSQL à utiliser, voir la partie *II. Installation* pour cela.
- “datasources” : On a ici les différentes sources de données que GDA utilise. On aura alors les différentes bases de données pour une source, les tables que contiennent ces bases, et les différentes versions de cette base (il peut y avoir une version pseudonymisée, k-anonymisée...) Prenons un exemple existant dans GDA, la base “banking”. Voilà la partie liée dans “datasources” :

On voit alors que chaque source de données doit avoir :

- Un “friendlyName”, qui permet d’afficher joliment son nom (inutile)
- Un tableau “tables”, qui représente les tables disponibles dans cette base

- Un tableau “databases”, qui répertorie toutes les bases représentant les versions de la source. Chaque base doit posséder au moins une version **raw**, contenant les données en brut. Dans notre exemple, la base *raw\_banking* contient les données de chaque table en brut.

- “anonClasses” : Cette partie a pour but de référencer toutes les techniques d’anonymisation, et les bases qui y sont associées.

Prenons un exemple disponible avec GDA de base, la k-anonymisation :

```

"K-anonymization": {
  "friendlyName": "K-anonymization",
  "naive": {
    "friendlyName": "Naive Configuration",
    "implementation": "ARX",
    "k_2": {
      "friendlyName": "K = 2",
      "service": "postgres",
      "databases": [
        "k_anon_2_banking_full",
        "k_anon_2_scihub_full",
        "k_anon_2_taxi_full",
        "k_anon_2_census_full"
      ]
    },
    "k_5": {
      "friendlyName": "K = 5",
      "service": "postgres",
      "databases": [
        "k_anon_5_banking_full",
        "k_anon_5_scihub_full",
        "k_anon_5_taxi_full",
        "k_anon_5_census_full"
      ]
    }
  }
},

```

On voit qu’une classe d’anonymisation possède :

- Un “friendlyName”, pour l’affichage (inutile)
- Des sous-parties de la technique, ici la seule sous-partie est la partie “naive”. Cette sous-partie a son tour peut avoir des sous-parties (ici k\_2 et k\_5), etc...

En fait on considère la partie “k\_anonymisation”, la racine de la technique, également comme de type sous-partie, qui peut contenir directement les informations d’une sous-partie. On pourrait alors avoir quelque chose de la forme suivante :

```

“anonClasses” : {
  “k_anonymisation” : {
    “friendlyName” : “blabla”,
    “service” : “postgres”,
    “databases” : [“k_anon_banking”, “k_anon_scihub”, “k_anon_census”...]
  },
  ...
}

```

Ceci étant la forme la plus courte pour définir une technique d’anonymisation, sans sous-partie.

Une sous-partie contient alors :

- Un “friendlyName”, pour l’affichage
- Un “service”, qui indique où se trouve la base, ici dans postgres. Dans notre cas avec des nouvelles bases, on utilisera postgres car c’est un serveur postgres qui est déployé.
- Un tableau “databases”, qui indique toutes les bases qui utilisent cette technique d’anonymisation. Dans notre cas, ici, nous devons rajouter les bases k-anonymisées.

- OPTIONNEL, deux tableaux de la même forme que “*databases*” :
  - ‘pubDatabases’ : Correspond à des databases qui seraient publiques (ex. besoin dans attaque linkability)
  - ‘linkDatabases’ : Correspond à des databases servant à établir un lien avec une autre (ex. besoin dans attaque linkability)
- Autant de sous-partie que nécessaire

Dans une attaque, la config pour utiliser par exemple une base anonymisée en *k\_5* anonymat ressemblera à cela, on retrouve bien les sous-parties de la technique renseignée dans le *master.json* :

```
config = {
  "configVersion": "compact1",
  "basic": {
    "attackType": "Test Attack",
    "criteria": "singlingOut"
  },
  'anonTypes': [
    ["k_anonymisation", "naive", "k_5"]
  ],
  'tables': [ ['banking', 'accounts'] ]
}
```

### IMPORTANT :

Il y a une technique appelée “no\_anon” dans cette partie des “anonClasses”, cela correspond aux bases de données contenant les données brutes, non anonymisées. Une nouvelle source de données **doit forcément avoir une base en brut**, qui doit être référencée ici. (voir *master.json.template* sur le Github)

#### B. Un *master.json* par source de données

Le but sur Github sera d’avoir autant de fichiers *master.json* que de sources de données, que les utilisateurs peuvent récupérer afin de pouvoir utiliser les bases. Si la source de données s’appelle “*voitures*”, vous trouverez un fichier *master.json.voitures* sur Github. Cela évitera d’avoir un énorme *master.json* commun à éditer chaque fois qu’une nouvelle source existe, évitant de potentielles erreurs ce qui rendrait le seul *master.json* incorrect.

### C. Créer un master.json pour votre nouvelle source de données

Lorsque vous créez une nouvelle source de données, donc avec des bases sur le serveur PostgreSQL, il vous faudra alors créer un *master.json* spécifique à votre source de donnée. Référez-vous à la partie A. sur la structure du fichier pour bien le comprendre avant de vous lancer.

Pour ce faire, vous allez utiliser le *master.json.template* disponible sur le Github :

- 1) Copiez ce template dans votre dossier config de GDA (là où est le *master.json* de base)
- 2) Renommez le *master.json* de base en "*master.json.gda*" par exemple
- 3) Renommez le template copié en "*master.json*", afin que ce soit ce fichier pris en compte par GDA
- 4) Commencez à le modifier, pour qu'il corresponde à votre source de données (voir exemple sur le Github, dossier des masters)
  - Il y a la partie "datasources" à remplir
  - "anonClasses" doit être rempli avec les techniques d'anonymisation que vous utilisez pour votre source, contenant pour chacune la database utilisant cette technique.
  - Dans "anonClasses", la partie "no\_annon" où vous devez mettre votre base avec les données brutes (raw\_{nom\_source} par exemple)
- 5) Si ce n'est pas déjà fait, utilisez le script pour créer les bases et tables sur le serveur, puis les populer avec vos fichiers CSV
- 6) Testez avec des attaques basiques que vos bases et votre *master.json* sont bien créés et sans erreur.
- 7) Une fois que vous êtes sûrs de vous, vous pouvez upload sur le Github votre *master.json* :
  - Renommez votre *master.json* par "*master.json.{nom\_source}*", copiez le ensuite dans le dossier du Github "masters json", et vous pouvez push
  - Un groupe qui voudra utiliser votre source ira piocher dans le Github, et suivra les instructions du D. sur l'utilisation d'une source.

Pour vous aider, il y a un exemple sur le Github, dans le dossier "example". Il y a un .txt avec les informations sur la source à créer , et le *master.json* associé.

#### D. Utiliser une source de données

Pour utiliser une source de données, il faut que le *master.json* ait toutes les informations de cette source. Une source de données aura son *master.json.{nom\_source}* disponible sur le Github, dans le dossier “masters json”.

Récupérez le *master.json* relatif à la source que vous voulez utiliser, et à partir de là vous aurez deux choix pour l'utiliser :





- 1) Le premier choix consiste à garder un *master.json.{nom\_source}* par source de données, et de renommer en “*master.json*” celui de la source à utiliser.

Copiez alors le *master.json* récupéré du Github au même endroit où se situe votre config GDA (là où le *master.json* est installé de base).

Ainsi, lorsque vous souhaitez utiliser la nouvelle source, vous renommez le *master.json* actuellement utilisé en “*master.json.{source}*”, source étant la source qu'il représente, et vous renommez le *master.json.{nom\_source}* à utiliser en “*master.json*”. Ce sera donc ce fichier *master.json* que GDA prendra en compte.

Renommez le *master.json* de base (gda), en *master.json.gda* quand non utilisé.

Vous aurez une architecture comme cela dans le dossier de config de GDA :

Nom	Modifié le	type	taille
 master.json	22/12/2020 16:56	Fichier JSON	7 Ko
 master.json.banking	22/12/2020 16:56	Fichier BANKING	7 Ko
 master.json.gda	22/12/2020 16:56	Fichier GDA	7 Ko
 master.json.voitures	22/12/2020 16:56	Fichier VOITURES	7 Ko

Avantages : on s'y retrouve, chaque source a son propre master

Inconvénients : Si on veut changer de source d'attaque, on doit passer par un renommage de ces fichiers afin d'utiliser la bonne source

- 2) Le second choix consiste à utiliser un seul *master.json* pour toutes les nouvelles sources.

Le *master.json* de GDA ayant un host et port différent du serveur de l'INSA, nous allons garder un *master.json*, pour GDA, et un pour celui de l'INSA. Ils seront nommés “*master.json.gda*” ou “*master.json.insa*” si non utilisés.

Pour ce faire, récupérez le master d'une source de données sur le Github. Copiez-le dans le dossier de config de GDA (où il y a le



*master.json* de base). Renommez le master de GDA en "*master.json.gda*", et renommez le nouveau en "*master.json*".

L'idée ici sera de fusionner les *master.json* pour n'en avoir qu'un gros. A chaque fois que vous souhaitez utiliser une nouvelle source de données, il vous faut télécharger sur le Github le bon *master.json*, et ensuite, à la main, compléter votre *master.json* général, avec les informations de la nouvelle source. Cela signifie copier la partie "datasource" concernant la nouvelle source dans le *master.json* partie "datasource".

Également fusionner la partie "annonClasses", en mettant les bases de la nouvelle source au bon endroit dans le *master.json*.

Avantage : On utilise qu'un fichier *master.json*, cela évite d'avoir à changer de *master.json* en fonction du programme.

Inconvénient : Travail assez important pour fusionner les masters, avec possibilité de se tromper, si nombre de sources important, fichier très volumineux, difficile de s'y retrouver.

### **Conclusion :**

La solution 2 présente plus d'inconvénient que la solution 1), de façon générale **il est alors préférable d'utiliser la solution 1)**. Après, si la situation le permet, la solution 2) pourrait être plus avantageuse.

## IV. Gestion des bases de données

L'utilisation du script pour gérer des bases est optionnelle, il est possible de se connecter au serveur et gérer PostgreSQL en ligne de commande ou avec une interface quelconque.

**⚠ Cependant, il est nécessaire de bien lire la partie [III.C](#), car cette partie indique comment avoir un *master.json* qui reconnaîtra votre nouvelle base une fois mise en ligne.**

**Juste l'ajouter sur le serveur ne suffit pas, GDA ne pourra pas accéder aux nouvelles bases créées sans renseigner correctement ce fichier *master.json*.**

### A. Première utilisation

Pour utiliser le script, vous devrez installer :

- Python 3.9
- psycopg2

Clonez ensuite le dépôt github : [https://github.com/Alex2mars/gda\\_insa](https://github.com/Alex2mars/gda_insa)

Rendez-vous dans le dossier scripts. C'est dans ce dossier que sont stockés le script et quelques éléments utiles comme des exemples pour l'utiliser.

Le script se nomme avec grande imagination "script.py". Il a besoin à son lancement de 2 paramètres qui sont les chemins vers 2 fichiers de config :

- *config.json*, qui contient les données de connexion au serveur postgres
- *gestion.json* qui contient les actions à réaliser

Le fichier *config.json* contient plusieurs éléments :

- "host" -> adresse IP du serveur BDD hébergé à l'INSA
- "server\_port" -> port de ce serveur BDD
- "username" -> le nom d'utilisateur qui vous a été attribué
- "password" -> le mot de passe correspondant

Pour chacune des actions que vous souhaitez réaliser, l'exécution sera la même :

**python3.9 script.py config.json gestion.json**

Le fichier *config.json*, une fois configuré, n'est pas fait pour être modifié, seul le fichier *gestion.json* changera en fonction de ce que vous voulez faire.

### B. Créer une base

Créez le *master.json* qui sera lié à votre source (voir [III.D](#))).

Dans le fichier *gestion.json*, renseignez 2 éléments :

- action: "create\_db"
- db\_name: "NAME\_OF\_DB"

Par défaut le fichier *gestion.json* contient ces éléments

△ Une fois la base créée, aucun autre utilisateur n'aura accès à cette base, même en lecture. Pour changer cela, utilisez la commande SQL suivante dans votre database :

*GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonly;*

Les utilisateurs sont dans le groupe *readonly*, alors cette commande les autorisera tous à lire votre base. Mais si des nouvelles tables sont créées plus tard, il faudra réexécuter la commande pour mettre à jour les droits sur les nouvelles tables.

△ Si une base est créée avec des majuscules, sachez qu'ensuite pour créer des tables, les populer, supprimer la base ou supprimer des tables, le nom devra être indiqué uniquement avec des minuscules, sinon elle ne sera pas trouvée.

### C. Créer des tables

Modifiez en premier lieu le document *master.json* de votre source de donnée (pour rajouter la table dans "datasource").

Dans le fichier *gestion.json*, renseignez 3 éléments :

- action: "create"
- db\_name: "NAME\_OF\_DB"
- tables\_to\_create: "SQL\_FILE"

Le fichier *SQL\_FILE* contient le code pour créer les tables. Vous avez des exemples pour créer des tables dans le dossier *samples*.

### D. Populer une base à partir d'un fichier

Le fichier *gestion.json* devra avoir la forme suivante :

- action: "populate"
- db\_name: "NAME\_OF\_DB"
- data\_to\_import: {'table\_name': 'file.csv', 'table\_2', 'file2.csv'}
- headers\_in\_csv: {'table\_name': true} (optionnel)

Le *data\_to\_import* est un objet indiquant le nom de la table et le fichier csv correspondant pour la populer.

Le paramètre *headers\_in\_csv* est optionnel. Il est à utiliser uniquement dans le cas où des headers sont utilisés dans un ou plusieurs fichiers csv, auquel

cas il faut le préciser dans l'objet. Vous trouverez des exemples dans le dossier samples.

### E. Supprimer des tables / une base

Modifiez le fichier *master.json* en conséquence (voir partie IIIa)).

Modifiez ensuite *gestion.json* de manière suivante :

- action: "delete"
- db\_name: "NAME\_OF\_DB"
- tables: ["table\_name", "table\_2"]

Pour supprimer la base :

- action: "delete\_db"
- db\_name: "NAME\_OF\_DB"