

Compte rendu (SAE)

Création d'une base de données



Sommaire:

- 2.1: Script manuel de création de la base de données.....
- 2.2: Modélisation et script de création avec <<AGL>>.....
- 2.3: Peuplement des tables

2.1) Script manuel de création de base de données (sur le logiciel DATA GRIP) :

1.

```
CREATE TABLE region (  
    region_code INTEGER PRIMARY KEY,  
    name VARCHAR  
);
```

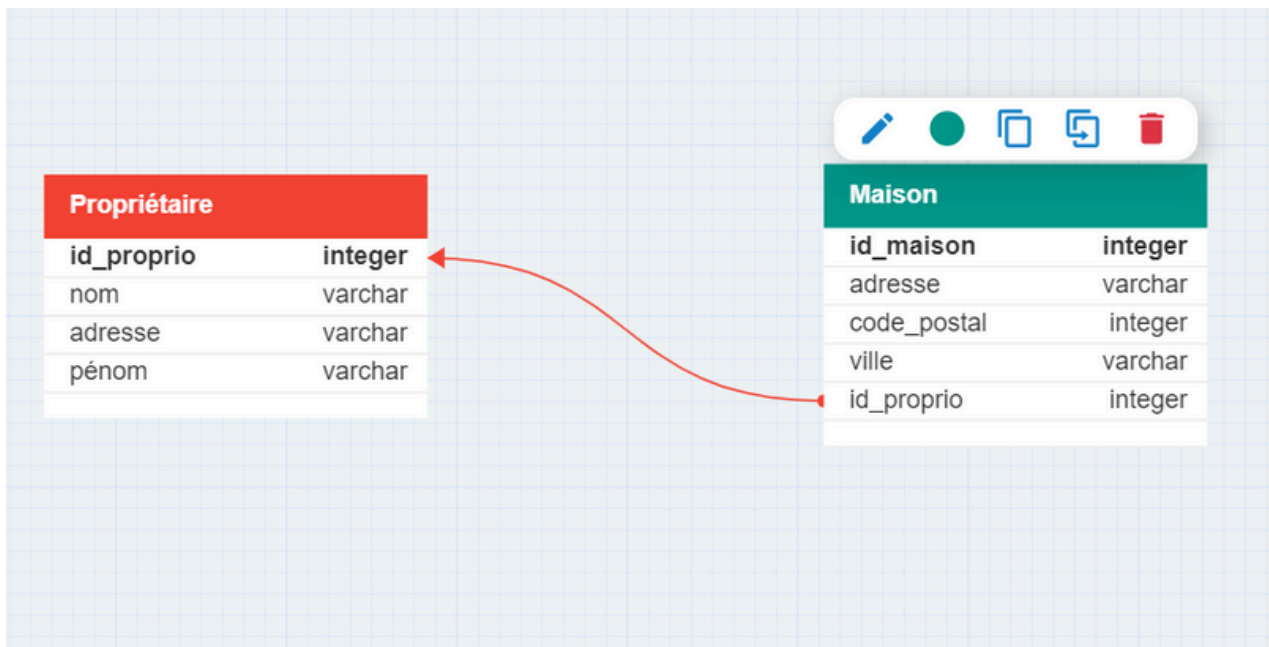
```
CREATE TABLE status (  
    status VARCHAR PRIMARY KEY  
);
```

```
CREATE TABLE country (  
    id_country SERIAL PRIMARY KEY,  
    name VARCHAR ,  
    region_code INTEGER REFERENCES region(region_code),  
    is_idc INTEGER  
);
```

```
CREATE TABLE freedom (  
    id_country INTEGER REFERENCES country(id_country),  
    year INTEGER ,  
    civil_liberties INTEGER ,  
    political_rights INTEGER,  
    status VARCHAR(255) REFERENCES status(status),  
    PRIMARY KEY (id_country, year)  
);
```

2.2) Modélisation et script de création << avec AGL >> :

1) Illustrations comparatives cours/AGL commentée d'une association fonctionnelle



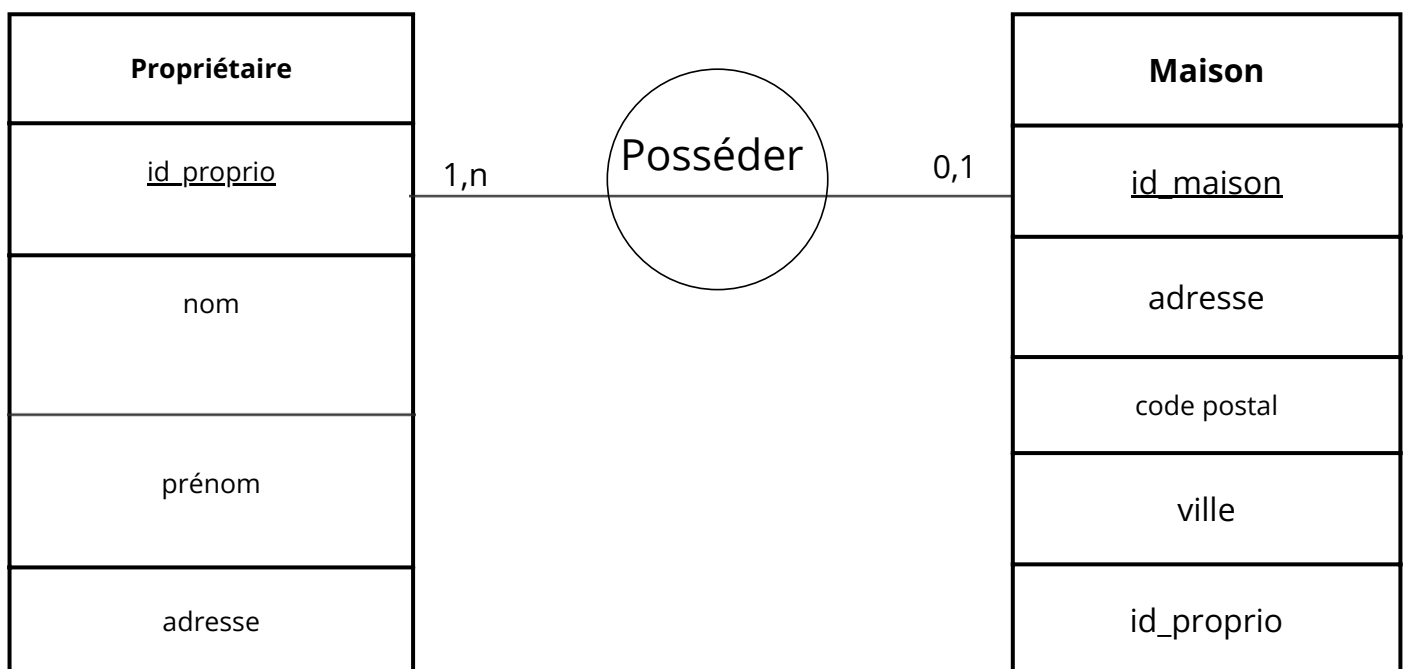
Sous la forme d'un type association fonctionnel:

Explication:

Une association **fonctionnelle** permet de décrire une relation entre **deux entités** dans une base de données. L'association fonctionnelle indique une **dépendance** fonctionnelle entre les attributs de deux entités. Plus précisément, elle signifie qu'une valeur particulière dans une entité détermine de manière unique une valeur dans une autre entité.

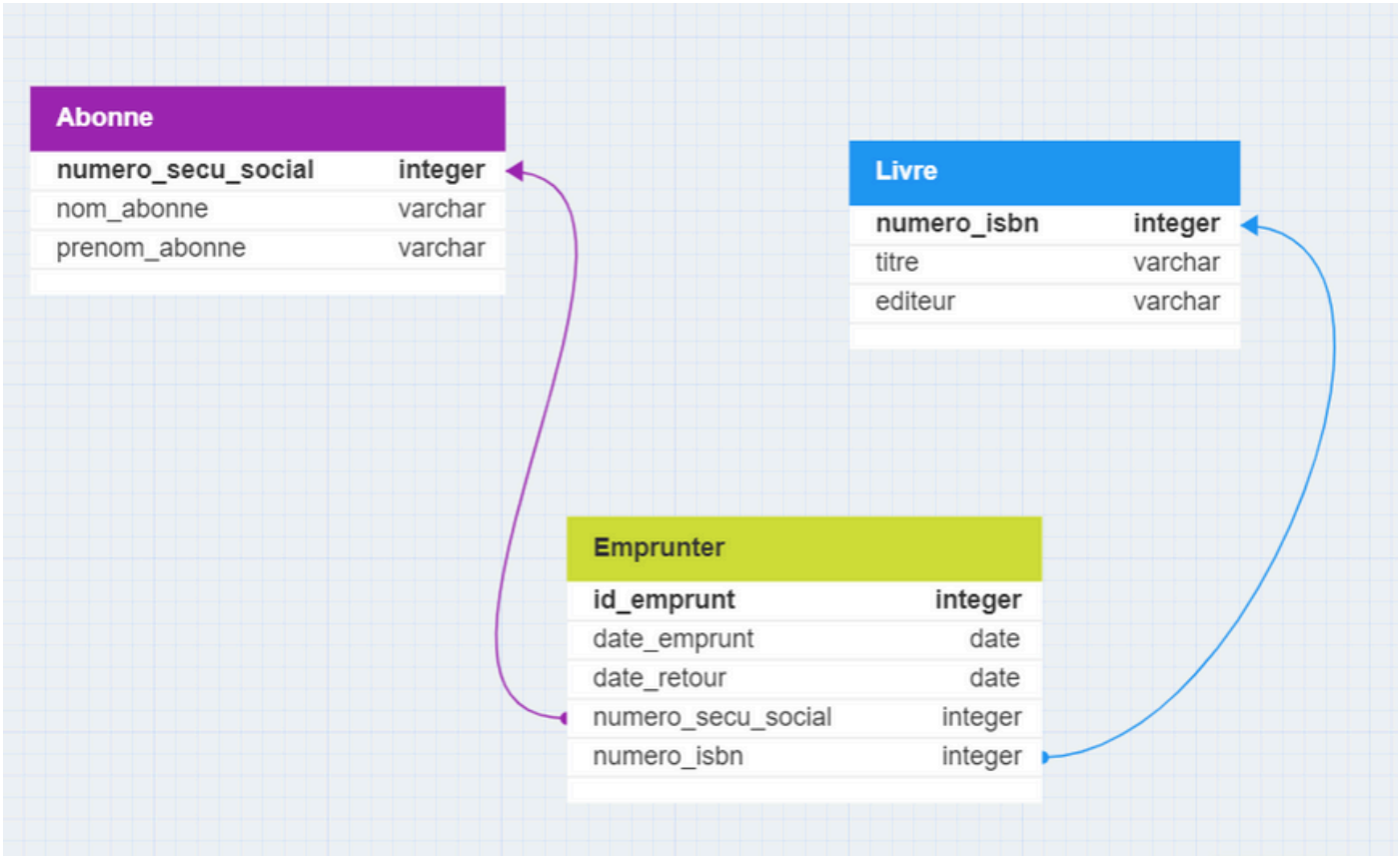
Supposons que nous ayons deux entités : "Propriétaire" et "Maison".

Chaque propriétaire peut posséder une ou plusieurs maisons (1:N), mais chaque maison ne peut être possédée que par un seul propriétaire ou ne pas avoir de propriétaire (0:1).



Nous constatons que la représentation sous la forme d'un type- association est plus claire et facile a comprendre pour un lecteur externe. Cependant la représentation sur l'AGL DB designer est plus concise et sera peut être plus adapter a quelqu'un qui a déjà des bases.

2) Illustrations comparatives cours/AGL commentée d'une association maillé

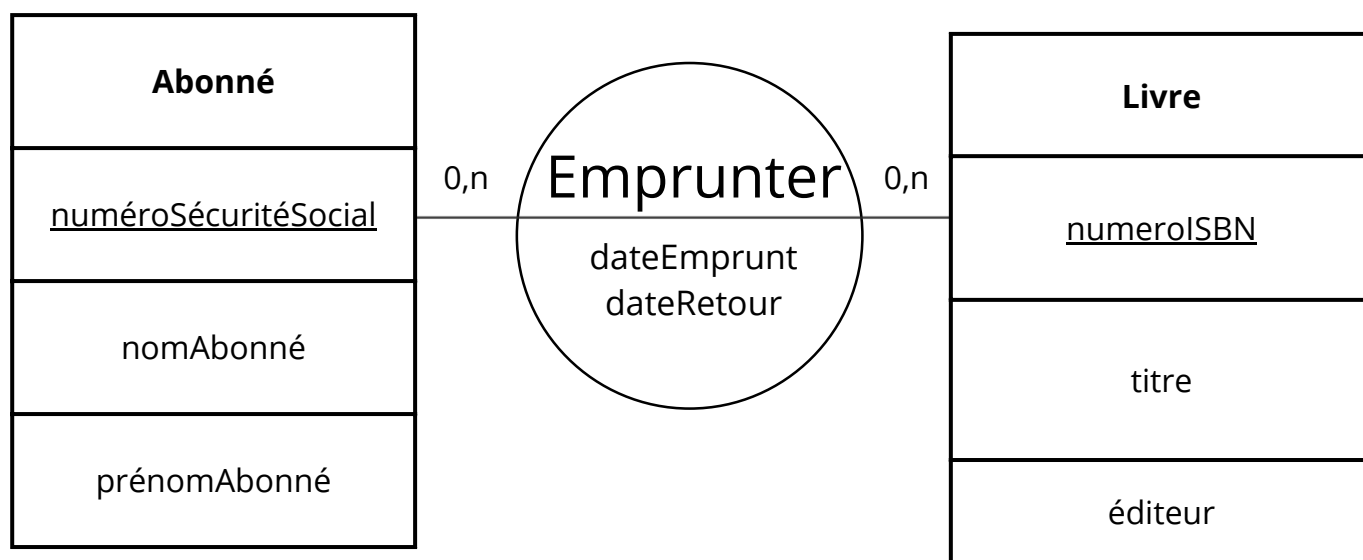


Sous la forme d'un type association maillé (exemple du cours):

Explication:

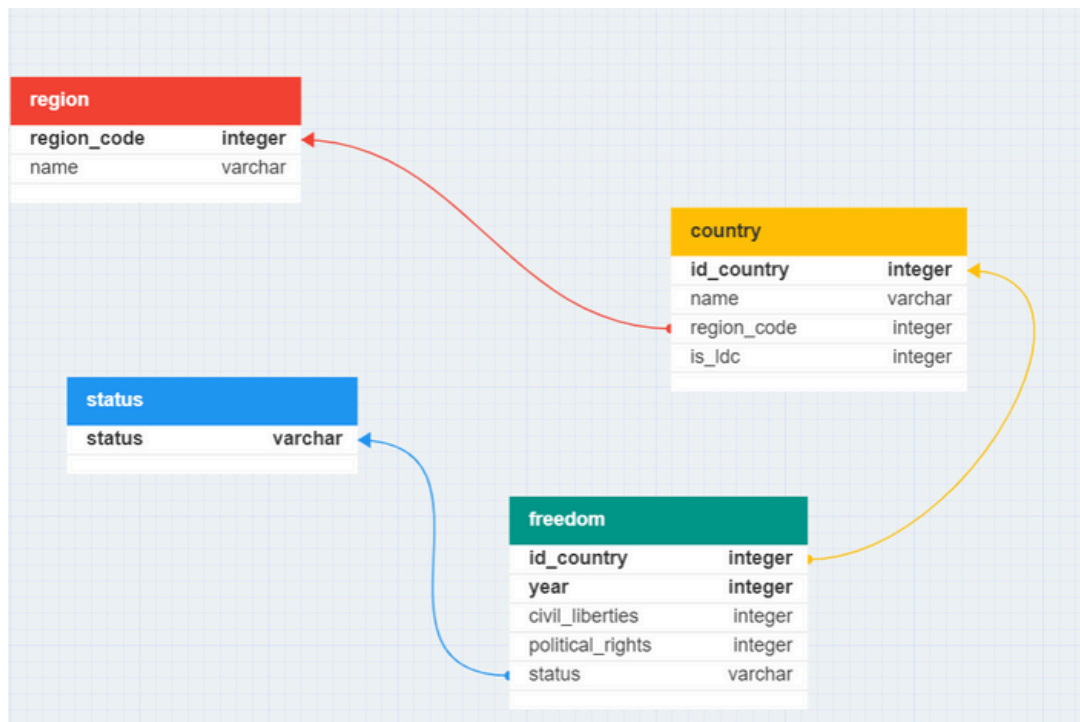
Une association **maillée** est une relation entre **deux entités** où **un événement** dans l'une peut être associé à **plusieurs événement** dans l'autre, et vice versa, sans nécessairement établir une correspondance unique.

Dans l'exemple ci-dessous un abonné peut emprunter 0 ou plusieurs livre et un livre peut être emprunter par 0 ou plusieurs abonnés



Comme nous l'avons dit lors de la première comparaison l'AGL et le représentation type association(ici maillé) ont chacun des avantages et des désavantages, tout dépend des besoins. Pour des relations simple nous préconisons le modèle type association mais pour des relations plus complexe nous pensons qu'il est préférable de se tourner vers l'AGL et ses nombreux outils qui permettent de générer automatiquement des scripts, de faciliter la manipulation des contraintes et donc d'éviter les erreurs .

3) Modèle physique de donnée réaliser avec l'AGL



4) Script SQL de création des tables généré automatiquement par l'AGL

```
region {
  region_code integer pk increments
  name varchar
}

status {
  status varchar pk increments
}

country {
  id_country integer pk increments
  name varchar
  region_code integer > region.region_code
  is_ldc integer
}

freedom {
  id_country integer pk increments > country.id_country
  year integer pk
  civil_liberties integer
  political_rights integer
  status varchar > status.status
}
```

5. Discussion sur les différences entre les scripts produits manuellement et automatiquement:

Les deux scripts ont visiblement la même structure, les principales différences que nous constatons résident dans leur syntaxe. On le voit par exemple dans la manière de faire appel aux clés étrangères, dans le script produit manuellement on utilise la clause "REFERENCES", pour le script généré automatiquement on voit que pour faire appel à une clé étrangère on utilise ce signe ">".

Les syntaxes de déclaration des clés primaires, des types de colonnes et des contraintes sont également différentes.

2.3) Peuplement des tables:

Pour réaliser le peuplement des tables nous avons d'abord créé une table temporaire "tmp" avec ce script:

```
CREATE TABLE tmp
(
  country      VARCHAR,
  year         INTEGER,
  civil_liberties INTEGER,
  political_right INTEGER,
  status       VARCHAR,
  region_code  INTEGER,
  region_name  VARCHAR,
  is_idc       INTEGER
);
```

Une fois la table temporaire créée, nous avons importé le fichier plat "freedom.csv" dedans en faisant un clic droit sur cette même table temporaire, puis nous avons sélectionné l'option "Import Data from File(s)".

Par la suite, après avoir importé toutes les données dans 'tmp', nous avons projeté les informations que contenait cette table dans les autres tables que nous avons créées en utilisant les instructions suivantes :

Pour la table **region**:

```
INSERT INTO region SELECT DISTINCT region_code, region_name FROM tmp;
```

Pour la table **status**:

```
INSERT INTO status SELECT DISTINCT status FROM tmp;
```

Pour la table **country**:

```
INSERT INTO country (name,region_code,is_ldc) SELECT DISTINCT country, region_code, is_ldc
FROM tmp;
```

Pour la table **freedom**:

```
INSERT INTO freedom (id_country, year, civil_liberties, political_rights, status)
SELECT country.id_country, tmp.year, tmp.civil_liberties, tmp.political_right, tmp.status
FROM tmp
JOIN country ON tmp.country = country.name;
```

Explication:

L'instruction **INSERT INTO** indique que l'on insère des données dans une table.

SELECT permet de sélectionner distinctement les valeurs des colonnes.

DISTINCT permet d'éviter les doublons

FROM montre qu'on importe ces valeurs depuis une autre tables.

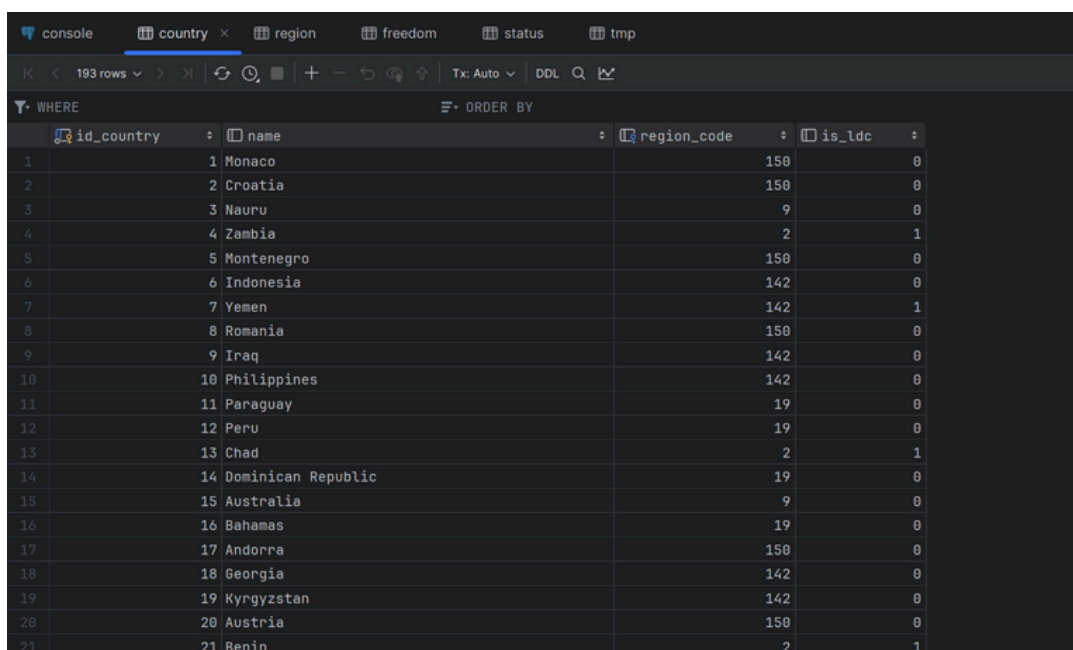
JOIN permet de combiner les données stockées dans différentes tables.

Pour la dernière table (freedom) il a fallut faire une jointure sur la condition que la colonne **country** de la table **tmp** soit égale à la colonne **name** de la table **country**.

Enfin pour vérifier toutes mes tables j'ai effectuer ces requêtes :

```
SELECT * FROM country;
SELECT * FROM region;
SELECT * FROM status;
SELECT * FROM freedom;
```

Résultat :



	id_country	name	region_code	is_ldc
1	1	Monaco	150	0
2	2	Croatia	150	0
3	3	Nauru	9	0
4	4	Zambia	2	1
5	5	Montenegro	150	0
6	6	Indonesia	142	0
7	7	Yemen	142	1
8	8	Romania	150	0
9	9	Iraq	142	0
10	10	Philippines	142	0
11	11	Paraguay	19	0
12	12	Peru	19	0
13	13	Chad	2	1
14	14	Dominican Republic	19	0
15	15	Australia	9	0
16	16	Bahamas	19	0
17	17	Andorra	150	0
18	18	Georgia	142	0
19	19	Kyrgyzstan	142	0
20	20	Austria	150	0
21	21	Benin	2	1

console country region x freedo

5 rows

WHERE

	region_code	name
1	142	Asia
2	9	Oceania
3	19	Americas
4	150	Europe
5	2	Africa

console country region freedom x status tmp

4 979 rows

WHERE ORDER BY

	id_country	year	civil_liberties	political_rights	status
1	62	1995	7	7	NF
2	62	1996	7	7	NF
3	62	1997	7	7	NF
4	62	1998	7	7	NF
5	62	1999	7	7	NF
6	62	2000	7	7	NF
7	62	2001	7	7	NF
8	62	2002	6	6	NF
9	62	2003	6	6	NF
10	62	2004	6	5	NF
11	62	2005	5	5	PF
12	62	2006	5	5	PF
13	62	2007	5	5	PF
14	62	2008	6	5	NF
15	62	2009	6	6	NF
16	62	2010	6	6	NF
17	62	2011	6	6	NF
18	62	2012	6	6	NF
19	62	2013	6	6	NF
20	62	2014	6	6	NF
21	62	2015	6	6	NF

console country region freedom status x

3 rows

WHERE ORDER BY

	status
1	PF
2	NF
3	F