Relationnel-JSON avec PostgreSQL

Table des matières

Cours

1. Introduction à la manipulation JSON sous PostgreSQL

1.1. Type JSON sous PostgreSQL

PostgreSQL propose un nouveau type de données qui permet d'insérer du JSON dans une table.

Il s'agit d'une solution pour coupler de l'imbrication NF^2 avec une base relationnelle.

Rappel

La syntaxe JSON en bref *(cf. p.15)*

Créer une table avec des attributs JSON — ***Syntaxe***

```
1 CREATE TABLE t (
2 ...
3 jsonatt JSON,
4 ...
5 );
6
```

Insérer du JSON dans une table — ***Syntaxe***

```
1 INSERT INTO t
2 VALUES (
3 ...
4 '{ "key1": "value1", "key2": "value2"}',
5 ...
6 );
```

Exemple

```
1 CREATE TABLE cours (
2 titre TEXT PRIMARY KEY,
3 auteur JSON NOT NULL,
4 chapitres JSON NOT NULL,
5 nbpages INTEGER
6 );
```

```
1 INSERT INTO cours (titre, auteur, chapitres, nbpages)
2 VALUES (
3 'Bases de données',
4 '{"nom":"Crozat","prenom":"Stéphane"}',
5 '["UML","R","SQL","Normalisation","RO"]',
6 98
7 );
8 INSERT INTO cours (titre, auteur, chapitres, nbpages)
```

```
9 VALUES (
10 'Ingénierie documentaire',
11 '{"nom":"Crozat","prenom":"Stéphane"}',
12 '["XML","JSON"]',
13 45
14 );
```

```
1           titre          |                auteur                |
chapitres                | nbpages
2-------------------------+--------------------------------------+------------------
----------------------+---------
3 Bases de données         | {"nom":"Crozat","prenom":"Stéphane"} |
["UML","R","SQL","Normalisation","RO"] |      98
4 Ingénierie documentaire | {"nom":"Crozat","prenom":"Stéphane"} | ["XML","JSON"]
|      45
```

Fondamental

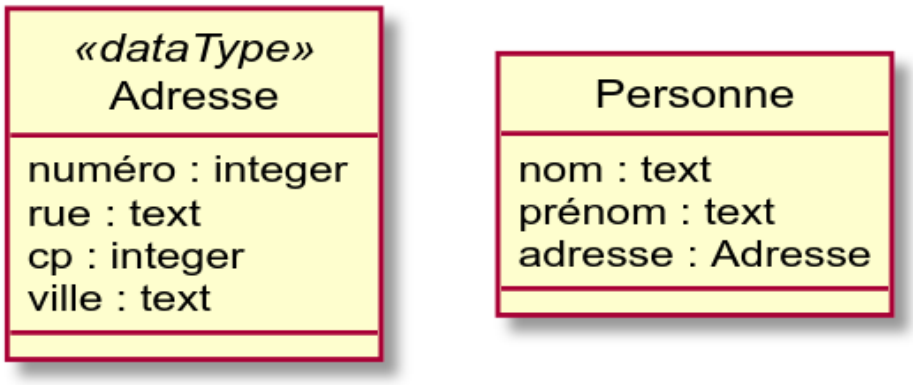
Pour manipuler les données JSON intégrées dans une base PosgreSQL, on dispose d'opérateur complémentaires permettant de projeter les représentations JSON en relationnel.

Par exemple :

- `->>` permet de projeter une valeur JSON
- `->` permet de parcourir un arbre JSON
- `JSON_ARRAY_ELEMENTS` permet de convertir un tableau de valeurs JSON en relationnel
- `JSON_TO_RECORDSET` permet de convertir un tableau d'objets JSON en relationnel

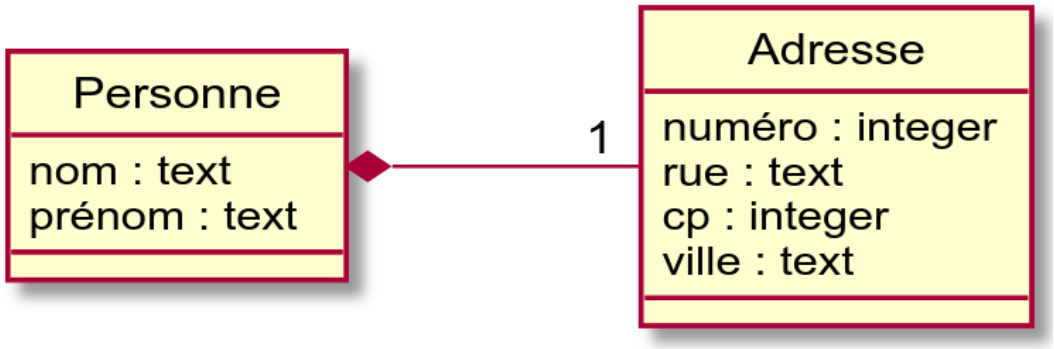
1.2. Attributs composés (datatype) en R-JSON (->>)

Exemple



Représentation équivalente

Rappel



Une représentation avec datatype est équivalente à une composition 1:1.

Exemple

```
CREATE TABLE Personne (
sk TEXT PRIMARY KEY,
nom TEXT NOT NULL,
prénom TEXT NOT NULL,
adresse JSON NOT NULL
);
```

```
INSERT INTO Personne
VALUES (
1,
'Holmes',
'Sherlock',
'{"numéro":221,"rue":"rue du boulanger", "cp":60200, "ville":"Compiègne"}'
);
```

```
 sk |  nom   |  prénom  |                                  adresse
----+--------+----------+-------------------------------------------------------------
----------------
 1  | Holmes | Sherlock | {"numéro":221,"rue":"rue du boulanger", "cp":60200,
"ville":"Compiègne"}
```

Obtenir une valeur d'un objet JSON : opérateur ->>('key')

Syntaxe

```
SELECT jsonatt->>'key' FROM t
```

Renvoie la valeur correspondant à la clé `key` située à la racine du JSON.

Exemple

```
SELECT nom, prénom, adresse->>'ville' AS ville
FROM Personne;
```

```
  nom   |  prénom  |   ville
--------+----------+-----------
 Holmes | Sherlock | Compiègne
```

Forcer les types (CAST)

Attention

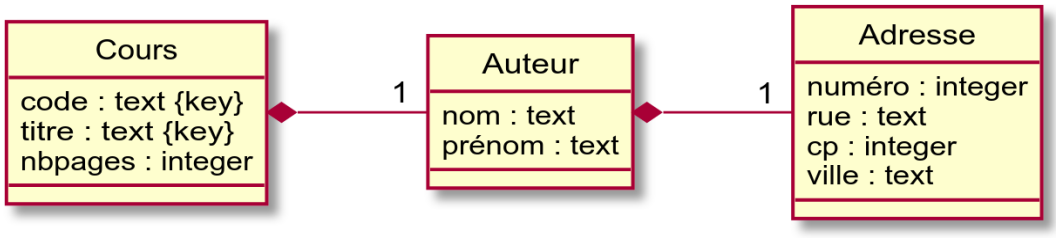
Les attributs sont retournés comme des chaînes, pour obtenir d'autre types, il faut utiliser la fonction CAST.

```
SELECT nom, prénom, CAST(adresse->>'numéro' AS INTEGER) AS numéro, adresse->>'rue'
AS rue
FROM Personne;
```

```
  nom   |  prénom  | numéro |       rue
--------+----------+--------+-----------------
 Holmes | Sherlock |    221 | rue du boulanger
```

1.3. Arborescences en R-JSON (->)

Exemple



Exemple

```sql
CREATE TABLE cours (
code TEXT PRIMARY KEY,
titre TEXT UNIQUE NOT NULL,
nbpages INTEGER NOT NULL,
auteur JSON NOT NULL
);
```

```sql
INSERT INTO cours (code, titre, nbpages, auteur)
VALUES (
'NF17',
'Bases de données',
98,
'{"nom":"Crozat","prenom":"Stéphane","adresse":{"num":18,"rue":"rue St Fiacre","cp":60200,"ville":"Compiègne"}}'
);
INSERT INTO cours (code, titre, nbpages, auteur)
VALUES (
'NF29',
'Ingénierie documentaire',
45,
'{"nom":"Crozat","prenom":"Stéphane","adresse":{"num":18,"rue":"rue St Fiacre","cp":60200,"ville":"Compiègne"}}'
);
```

```sql
INSERT INTO cours (code, titre, nbpages, auteur)
VALUES (
'NF29',
'Ingénierie documentaire',
45,
'{"nom":"Crozat","prenom":"Stéphane","adresse":{"num":4,"rue":"rue St Germain","cp":60300,"ville":"Senlis"}}'
);
```

Parcourir un JSON (opérateur ->)

Syntaxe

```sql
SELECT jsonatt->'key1'->>'key2' FROM t;
```

Renvoie la valeur associée à `key2` qui se trouve dans `key1`.

Exemple

```sql
SELECT
titre,
auteur->>'nom' AS nom_auteur,
auteur->>'prenom' AS prenom_auteur,
auteur->'adresse'->>'ville' AS ville_auteur
FROM cours;
```

```
          titre          | nom_auteur | prenom_auteur | ville_auteur
-------------------------+------------+---------------+--------------
 Bases de données        | Crozat     | Stéphane      | Compiègne
 Ingénierie documentaire | Crozat     | Stéphane      | Senlis
```

Attention

- -> Est un opérateur de parcours intermédiaire, qui renvoie du JSON
- ->> Est un opérateur final qui renvoie une valeur.

Opérateurs — **Complément**

Operator	Right Operand Type	Description	Example	Example Result
->	int	Get JSON array element (indexed from zero, negative integers count from the end)	'[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2	{"c":"baz"}
->	text	Get JSON object field by key	'{"a": {"b":"foo"}}'::json->'a'	{"b":"foo"}
->>	int	Get JSON array element as text	'[1,2,3]'::json->>2	3
->>	text	Get JSON object field as text	'{"a":1,"b":2}'::json->>'b'	2
#>	text[]	Get JSON object at specified path	'{"a": {"b":{"c": "foo"}}}'::json#>'{a,b}'	{"c": "foo"}
#>>	text[]	Get JSON object at specified path as text	'{"a":[1,2,3],"b":[4,5,6]}'::json#>>'{a,2}'	3

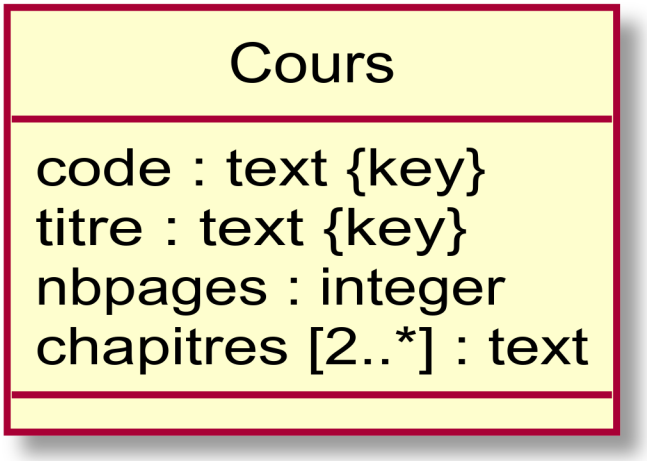
json Operators

Complément

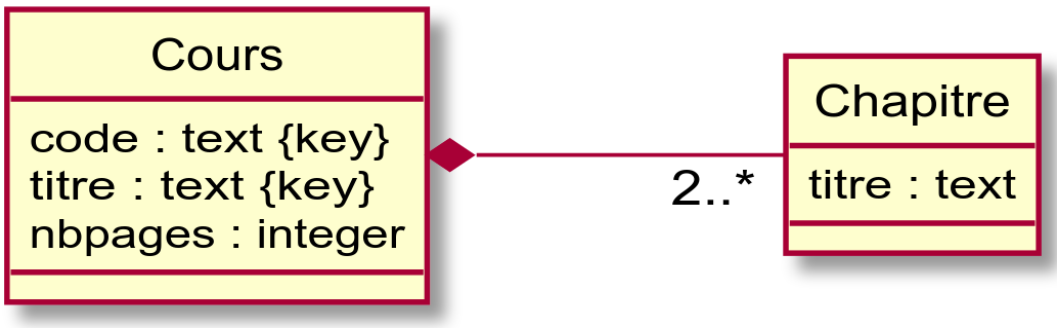
https://www.postgresql.org/docs/current/functions-json.html

1.4. Attributs multivalués en R-JSON (JSON_ARRAY_ELEMENTS)

Exemple



Représentation équivalente — **Rappel**



Une représentation avec attribut multivalué est équivalente à une composition dont le composite ne comporte qu'un seul attribut.

Exemple

```
CREATE TABLE cours (
code TEXT PRIMARY KEY,
titre TEXT UNIQUE NOT NULL,
nbpages INTEGER NOT NULL,
chapitres JSON NOT NULL
);
```

```
INSERT INTO cours (code, titre, nbpages, chapitres)
VALUES (
'NF17',
'Bases de données',
98,
'["UML","R","SQL","Normalisation","RO"]'
);
INSERT INTO cours (code, titre, nbpages, chapitres)
VALUES (
'NF29',
'Ingénierie documentaire',
45,
'["XML","JSON"]'
);
```

```
 code |          titre           | nbpages |               chapitres
------+--------------------------+---------+---------------------------------------
 NF17 | Bases de données         |      98 | ["UML","R","SQL","Normalisation","RO"]
 NF29 | Ingénierie documentaire  |      45 | ["XML","JSON"]
```

Convertir un tableau de scalaires JSON en relationnel : opérateur JSON_ARRAY_ELEMENTS

Syntaxe

```
SELECT a.* FROM t, JSON_ARRAY_ELEMENTS(t.jsonatt) a
```

Exemple

```
SELECT co.titre, ch.*
FROM cours co, JSON_ARRAY_ELEMENTS(co.chapitres) ch;
```

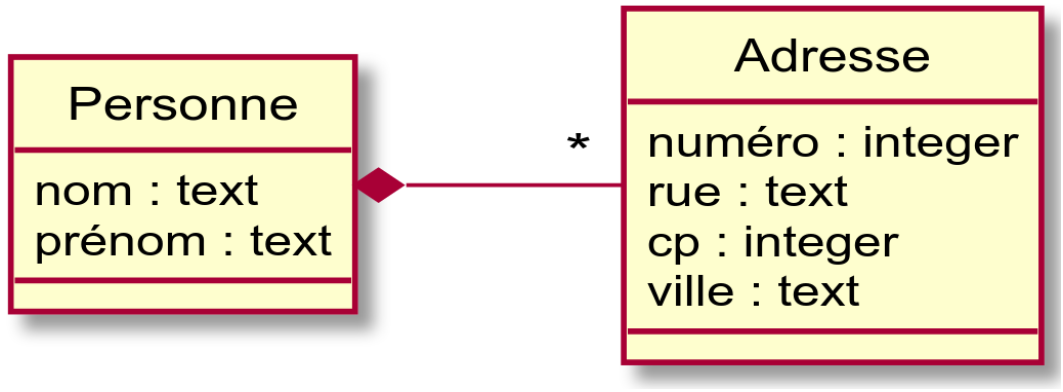
```
          titre          |      value
-------------------------+-----------------
 Bases de données        | "UML"
 Bases de données        | "R"
 Bases de données        | "SQL"
 Bases de données        | "Normalisation"
 Bases de données        | "RO"
 Ingénierie documentaire | "XML"
 Ingénierie documentaire | "JSON"
```

Complément

Complément : Attributs multivalués : opérateurs complémentaires *(cf. p.12)*

1.5. Compositions en R-JSON (JSON_ARRAY_ELEMENTS et ->>)

Exemple



Exemple

```
CREATE TABLE Personne (
sk TEXT PRIMARY KEY,
nom TEXT NOT NULL,
prénom TEXT NOT NULL,
adresse JSON
);
```

```
INSERT INTO Personne
VALUES (
1,
'Holmes',
'Sherlock',
'[
{"numéro":221,"rue":"rue du boulanger", "cp":60200, "ville":"Compiègne"},
{"numéro":0,"rue":"rue Secrète", "cp":60200, "ville":"Compiègne"}
]'
);

INSERT INTO Personne
VALUES (
2,
'Watson',
'John',
'[
{"numéro":221,"rue":"rue du boulanger", "cp":60200, "ville":"Compiègne"}
]'
);
```

```
 sk |  nom   |  prénom  |                                  adresse
----+--------+----------+-------------------------------------------------------------
-----------------
 1  | Holmes | Sherlock | [
+
    |        |          | {"numéro":221,"rue":"rue du boulanger", "cp":60200,
"ville":"Compiègne"},+
    |        |          | {"numéro":0,"rue":"rue Secrète", "cp":60200,
"ville":"Compiègne"}        +
    |        |          | ]
 2  | Watson | John     | [
+
    |        |          | {"numéro":221,"rue":"rue du boulanger", "cp":60200,
"ville":"Compiègne"} +
    |        |          | ]
```

Convertir un tableau d'objets JSON en relationnel : opérateurs JSON_ARRAY_ELEMENTS et ->>

Syntaxe

```
SELECT a->>a1, a->>a2... FROM t, JSON_ARRAY_ELEMENTS(t.jsonatt) a
```

Exemple

```
SELECT p.nom, p.prénom, CAST(ad->>'numéro' AS INTEGER) AS numéro, ad->>'rue' AS
rue, ad->>'ville' AS ville
FROM personne p, JSON_ARRAY_ELEMENTS(p.adresse) ad
```

```
  nom   |  prénom  | numéro |       rue        |   ville
--------+----------+--------+------------------+-----------
 Holmes | Sherlock |    221 | rue du boulanger | Compiègne
 Holmes | Sherlock |      0 | rue Secrète      | Compiègne
 Watson | John     |    221 | rue du boulanger | Compiègne
```

Rappel

Il est nécessaire d'utiliser la fonction CAST pour obtenir un type autre que chaîne de caractère.

Attributs composés (datatype) en R-JSON (->>) *(cf. p.4)*

Complément

Complément : Compositions avec JSON_TO_RECORDSET *(cf. p.11)*

2. Éléments complémentaires pour la manipulation des compositions

2.1. Optionnalité avec JSON_ARRAY_ELEMENTS

LEFT JOIN JSON_ARRAY_ELEMENTS

Exemple

```
INSERT INTO Personne
VALUES (
1,
'Holmes',
'Sherlock',
'[
{"numéro":221,"rue":"rue du boulanger", "cp":60200, "ville":"Compiègne"},
{"numéro":0,"rue":"rue Secrète", "cp":60200, "ville":"Compiègne"}
]'
);

INSERT INTO Personne
VALUES (
2,
'Watson',
'John'
);
```

```
 sk |  nom   |  prénom  |                                    adresse
----+--------+----------+-----------------------------------------------------------------
-----------------
 1  | Holmes | Sherlock | [
+
    |        |          | {"numéro":221,"rue":"rue du boulanger", "cp":60200,
"ville":"Compiègne"},+
```

```
    |        |          | {"numéro":0,"rue":"rue Secrète", "cp":60200,
"ville":"Compiègne"}        +
    |        |          | ]
2   | Watson | John     |
```

Remarque

```
SELECT a->>a1, a->>a2... FROM t, JSON_ARRAY_ELEMENTS(t.jsonatt) a
```

équivalent à

```
SELECT a->>a1, a->>a2... FROM t JOIN JSON_ARRAY_ELEMENTS(t.jsonatt) a ON TRUE
```

L'opérateur JSON_ARRAY_ELEMENTS agit comme une jointure

Attention

```
SELECT p.nom, p.prénom, CAST(ad->>'numéro' AS INTEGER) AS numéro, ad->>'rue' AS
rue, ad->>'ville' AS ville
FROM personne p JOIN JSON_ARRAY_ELEMENTS(p.adresse) ad ON TRUE
```

```
  nom   |  prénom  | numéro |       rue        |   ville
--------+----------+--------+------------------+-----------
 Holmes | Sherlock |    221 | rue du boulanger | Compiègne
 Holmes | Sherlock |      0 | rue Secrète      | Compiègne
```

Convertir un tableau optionnel d'objets JSON en relationnel : opérateurs LEFT JOIN JSON_ARRAY_ELEMENTS et ->>

Syntaxe

```
SELECT a->>a1, a->>a2... FROM t LEFT JOIN JSON_ARRAY_ELEMENTS(jsonatt) a on TRUE
```

L'opérateur JSON_ARRAY_ELEMENTS agit comme une jointure

Exemple

```
SELECT p.nom, p.prénom, CAST(ad->>'numéro' AS INTEGER) AS numéro, ad->>'rue' AS
rue, ad->>'ville' AS ville
FROM personne p LEFT JOIN JSON_ARRAY_ELEMENTS(p.adresse) ad ON TRUE
```

```
  nom   |  prénom  | numéro |       rue        |   ville
--------+----------+--------+------------------+-----------
 Holmes | Sherlock |    221 | rue du boulanger | Compiègne
 Holmes | Sherlock |      0 | rue Secrète      | Compiègne
 Watson | John     |        |                  |
```

2.2. Complément : Compositions avec JSON_TO_RECORDSET

Convertir un tableau d'objets JSON en relationnel : opérateur JSON_TO_RECORDSET (à partir de Postgres 11)

Complément

```
SELECT a.* FROM JSON_TO_RECORDSET(jsonatt) a (a1 type1, a2, type2...)
```

Les attributs JSON à projeter en attribut relationnel doivent être explicitement mentionnés et typés.

```
SELECT p.nom, p.prénom, ad.*
FROM personne p, JSON_TO_RECORDSET(adresse) ad (numéro INTEGER, rue TEXT, ville
TEXT);
```

```
  nom   |  prénom  | numéro |       rue        |   ville
--------+----------+--------+------------------+-----------
 Holmes | Sherlock |    221 | rue du boulanger | Compiègne
 Holmes | Sherlock |      0 | rue Secrète      | Compiègne
 Watson | John     |    221 | rue du boulanger | Compiègne
```

Complément

https://www.postgresql.org/docs/current/functions-json.html

2.3. Complément : Opérateurs complémentaires pour les tableaux JSON

Obtenir une valeur d'un tableau JSON : opérateur ->>(integer)  ***Complément***

```
SELECT jsonatt->>N FROM t
```

Renvoie le Nième élément du tableau situé à la racine du JSON (en comptant à partir de 0).

```
SELECT
titre,
chapitres->>0 AS chapitre1
FROM cours;
```

```
          titre          | chapitre1
-------------------------+-----------
 Bases de données        | UML
 Ingénierie documentaire | XML
```

Combinaison arborescence et tableau  ***Complément***

```
SELECT jsonatt->'key1'->>0 FROM t;
```

Renvoie la première valeur du tableau associé dans `key1`.

```
INSERT INTO cours (code, titre, nbpages, chapitres)
VALUES (
'NF26',
'Datawarehouses',
67,
'{"sections":["Principes","Modélisation","ETL"]}'
);

SELECT
titre,
chapitres->'sections'->>0 AS chapitre1
FROM cours;
```

```
          titre          | chapitre1
-------------------------+-----------
 Bases de données        |
 Ingénierie documentaire |
 Datawarehouses          | Principes
```

 ***Complément***

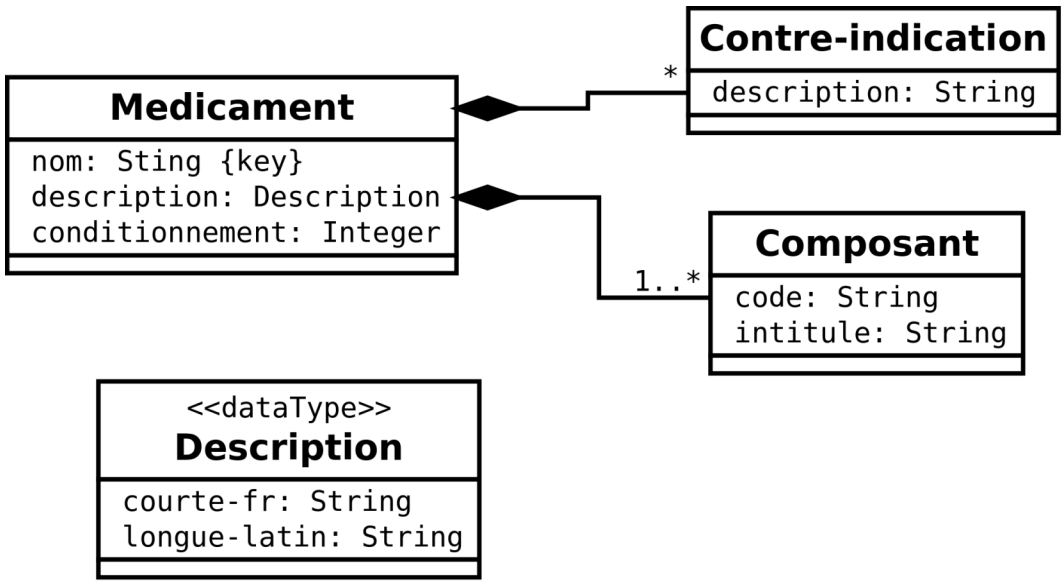
https://www.postgresql.org/docs/current/functions-json.html

Exercice

1. Exercice : Lab VIII

30 min

Soit le modèle UML ci-après.




```
Le Chourix a pour description courte "Médicament contre la chute des choux" et pour description longue "Vivamus fermentum semper porta. Nunc diam velit, adipiscing ut tristique vitae, sagittis vel odio. Maecenas convallis ullamcorper ultricies. Curabitur ornare.". Il est conditionné en boîte de 13.
Ses contre-indications sont :
- Le Chourix ne doit jamais être pris après minuit.
- Le Chourix ne doit jamais être mis au contact avec de l'eau.
Ses composants sont le HG79 ("Vif-argent allégé") et le SN50 ("Pur étain").

Le Tropas a pour description courte "Médicament contre les dysfonctionnements intellectuels" et pour description longue "Suspendisse lectus leo, consectetur in tempor sit amet, placerat quis neque. Etiam luctus porttitor lorem, sed suscipit est rutrum non.". Il est conditionné en boîte de 42.
Ses contre-indications sont :
- Le Tropas doit être gardé à l'abri de la lumière du soleil.
Son unique composant est le HG79 ("Vif-argent allégé").
```

Question 1

Réaliser l'implémentation R-JSON de ce modèle UML et insérer des données d'exemple.

Question 2

Proposez 3 vues permettant de visualkiser :

- La liste des noms des médicaments avec leurs contre-indications
- La liste des des noms des médicaments avec leurs composants
- La liste des noms des médicaments avec leur description courte, leur conditionnement, le nombre de leurs contre-indications et le nombre de leurs composants

Contenus annexes

1. La syntaxe JSON en bref

Règles syntaxiques *Syntaxe*

- Il ne doit exister qu'un seul élément père par document contenant tous les autres : **un élément racine**.
- Tout **fichier JSON bien formé** doit être :
  - **soit un objet** commençant par { et se terminant par },
  - **soit un tableau** commençant par [et terminant par].

  Cependant ils peuvent être vides, ainsi [] et {} sont des JSON valides.
- Les **séparateurs** utilisés entre deux paires/valeurs sont des **virgules**.
- Un objet JSON peut contenir d'autres objets JSON.
- Il ne peut pas y avoir d'éléments croisés.

Éléments du format JSON *Fondamental*

Il existe deux types d'éléments :

- Des couples de type **"nom": valeur**, comme l'on peut en trouver dans les tableaux associatifs.
- Des listes de valeurs, comme les tableaux utilisés en programmation.

Valeurs possibles *Définition*

- Primitifs : nombre, booléen, chaîne de caractères, null.
- Tableaux : liste de valeurs (tableaux et objets aussi autorisés) entrées entre crochets, séparées par des virgules.
- Objets : listes de couples "nom": valeur (tableaux et objets aussi autorisés) entrés entre accolades, séparés par des virgules.

Exemple

```
{
   "nom cours" : "NF29",
   "theme" : "ingenierie documentaire",
   "etudiants" : [
                 {
                   "nom" : "Norris",
                   "prenom" : "Chuck",
                   "age" : 73,
                   "pays" : "USA"
                 },
```

```
                {
                  "nom" : "Doe",
                  "prenom" : "Jane",
                  "age" : 45,
                  "pays" : "Angleterre"
                },
                {
                  "nom" : "Ourson",
                  "prenom" : "Winnie",
                  "age" : 10,
                  "pays" : "France"
                }
              ]
}
```

Index