Introduction aux bases de données NoSQL

Table des matières

Contexte

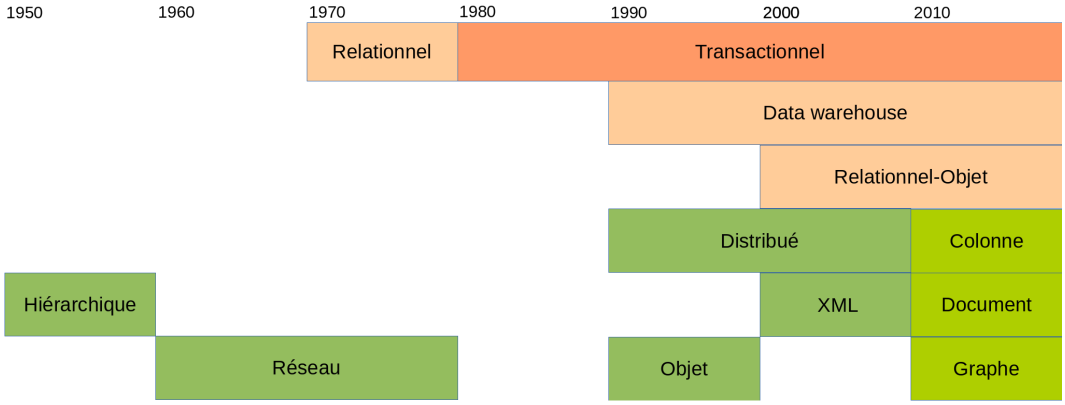
Durée : 2h

Environnement de travail : Repl.it

Pré-requis : Aucun

[cf. rOdgk83R]

Le modèle relationnel n'a pas été le seul et unique modèle des bases de données : différents autres modèles et technologies ont été explorés au cours des années. À la naissance de l'informatique, plusieurs modèles de stockage de l'information sont explorés, comme les modèles hiérarchique ou réseau.



Historique des technologies des bases de données

C'est néanmoins le modèle relationnel qui l'emporte dans les années 1970 pour sa simplicité et sa rigueur. Ce modèle présente ce qu'il y a de mieux pour assurer le contrôle de l'intégrité des données.

Depuis l'essor de l'informatisation dans les années 1990 puis celui du Web dans les années 2000, la question de la gestion des données se pose différemment. D'une part la question de la cohérence n'est plus de premier plan dans tous les contexte de gestion des données, par exemple lors de l'étude statistique de données ou lors de la gestion de contenus non critiques. D'autre part de plus en plus d'applications requièrent la gestion de très grandes quantités d'informations, fortement liées entre elles, pour lesquelles la représentation relationnelle n'est plus adaptée. Cette inadéquation pose des problèmes de performance, par exemple pour la gestion des réseaux sociaux.

Ces limitations des bases de données relationnelles ont amené un retour des modèles alternatifs au modèle relationnel. On parle du mouvement NoSQL, pour *Not Only SQL*. Les bases de données NoSQL remettent en cause l'hégémonie des SGBDR telle qu'elle s'est constituée dans les années 1980 en proposant des concepts empruntés aux représentations hiérarchique ou réseau proposées dans les années 1960.

Nous revenons dans ce module sur les problèmes rencontrés avec les bases de données relationnelles, pour ensuite présenter les alternatives proposées par le mouvement NoSQL.

Fondements du relationnel

[cf. PJoNQrhU]

Mise en situation

Les bases de données dites **relationnelles** sont les plus connues et les plus utilisées.

Mais au delà du simple fait de stocker des données, ces bases de données répondent à des contraintes fortes, dénotées par le mot **relationnel**.

Réfléchissez à ce qu'il se passe lorsque vous insérez des données en base : elles doivent respecter une structure pré-définie, celle des tables, ainsi que des contraintes, comme l'unicité de la clé primaire.

Aussi, un attribut ne doit pas contenir plusieurs informations : on sépare assez naturellement le nom, le prénom et la date de naissance dans une table.

Tous ces réflexes sont une conséquence des fondements théoriques du modèle relationnel.

Fonction première de toute base de données : stocker et retrouver

 *Fondamental*

La première fonction d'une base de données est de permettre de **stocker** et **retrouver** l'information.

Fonction secondaire d'une base de données relationnelle : la cohérence des données

 *Fondamental*

Une base de données relationnelle assure la **cohérence** des information relativement à un **schéma** exprimé a priori et ce quelles que soient ses conditions d'utilisation.

Une base de données n'est pas seulement un outil pour stocker et retrouver de l'information, c'est aussi un outil permettant d'exprimer un ensemble de règles relatives à ces données (type, format, optionnalité, unicité, références, etc.) et assurant le respect de ces règles **en premier lieu**.

Corollaire

 *Remarque*

1. Une base de données relationnelle bien conçue est toujours cohérente par rapport à son schéma.

   C'est donc un outil idéal pour réaliser des applications pour lesquelles l'intégrité des données est non négociable, par exemple pourune application de transaction bancaire.

2. Rien ne peut outrepasser le respect des règles de cohérence. La préservation de l'intégrité sera toujours préférée à la rapidité d'exécution de la fonction de stockage ou de recherche.

   C'est donc un outil qui peut s'avérer contre-productif pour des applications pour lesquelles la cohérence n'est pas indispensable mais où les performances le sont, par exemple pour un outil de recherche documentaire.

Relationnel et contrôle de l'intégrité des données

Rappel

Le relationnel propose un modèle théorique puissant et simple pour gérer le contrôle et l'intégrité des données. On notera en particulier les notions suivantes :

- L'**identification par les données**
- L'**atomicité**
- Le **schéma**
- La **non-redondance**

Identification par les données

Définition

Toute relation contient au moins une clé (un ensemble de données unique et non nul). La clé permet d'identifier un enregistrement. Il n'existe aucune autre technique pour identifier une donnée.

Cette règle de conception est issue de la première forme normale (1NF). Elle assure l'indépendance entre les données et le système physique de stockage des données. Si l'on porte les données d'un système à un autre, aucune information n'est perdue.

Atomicité

Définition

Les informations doivent être découpées afin que chaque case d'une relation ne contiennent qu'une seule donnée à la fois. Il n'existe qu'une structure de données, la relation. Une relation est un tableau dont toutes les lignes et colonnes sont identiques en termes de structure.

Cette règle est issue de la première forme normale (1NF). Elle assure que les données sont toujours accessible à travers le formalisme de l'algèbre relationnelle et donc qu'une requête SQL simple permet toujours de trouver le niveau d'information requis. Par exemple si on sépare bien le nom et le prénom dans deux colonnes séparée, on a pas besoin de fonction de traitement pour extraire l'un ou l'autre et on ne risque pas de se tromper (inversion nom et prénom, prénom composé, etc.).

Schéma

Définition

Il est nécessaire d'exprimer les règles de cohérence concernant les données *a priori*.

L'expression de ces règles permet de déléguer leur contrôle au système. Ainsi une base de données relationnelle est assurée d'être, en toutes circonstances, cohérente relativement à son schéma.

Non-redondance

Définition

Toute information n'est inscrite qu'une seule fois dans la base de données. On utilise un principe de décomposition pour éviter la redondance (l'information est découpée au sein de plusieurs tables). On utilise l'opération de jointure pour retrouver l'information consolidée.

Cette règle est issue de la théorie de la normalisation relationnelle (et en particulier de la troisième forme normale ou 3NF). Elle permet de faire en sorte que la base ne contiennent jamais d'information contradictoire. En effet si on autorise une données à être inscrite plusieurs fois alors il arrivera que des inscriptions soient contradictoire : la donnée aura plusieurs valeurs différentes. Dans ce cas précis, on ne peut plus décider quelle valeur est la bonne, ni même si l'une des valeurs inscrites est la bonne : l'information est perdue.

[cf. JCwNysyh]

Exercice : Appliquer la notion

Soit la base de données suivante représentant des recettes de cuisine :

```sql
CREATE TABLE recipe (
name VARCHAR(255),
ingredient VARCHAR(255) NOT NULL,
PRIMARY KEY (name, ingredient)
);

INSERT INTO recipe VALUES (
'Mayonnaise',
'2, cl, huile'
);
INSERT INTO recipe VALUES (
'Mayonnaise',
'1, cl, vinaigre'
);
INSERT INTO recipe VALUES (
'Mayonnaise',
'1, cl, moutarde'
);
```

```
    name     |   ingredient
-------------+-----------------
 Mayonnaise | 2, cl, huile
 Mayonnaise | 1, cl, vinaigre
 Mayonnaise | 1, cl, moutarde
```

Question 1

La base n'est pas en première forme normale car un de ses attributs n'est pas atomique, lequel ?

Question 2

Décomposer cet attribut en trois attributs pour respecter la règle d'atomicité.

Stéphane Crozat, Julien Jerphanion (Contributions : Ulysse Meyer, Amadou Sonko, Harold Carrel-Billiard, Maxime Uzan, Cécile Fecherolle)

Fondamentaux du non-relationnel

[cf. mVqO5IeC]

Mise en situation

Les bases de données relationnelles se basent sur des fondements théoriques forts, qui permettent par exemple d'assurer la cohérence des données selon une structure pré-définie.

Mais cette garantie sur la cohérence des données vient aussi avec des implications au niveau de la performance : plus les contraintes sont fortes, plus l'opération de stockage demande du temps.

Or, il y a des cas où le volume de données à gérer est massif, comme pour un moteur de recherche. A-t-on vraiment besoin que les résultats soient absolument cohérents dans ce cas ? Qu'il n'y ait pas de doublons ? Pas nécessairement, mais en revanche, le plus important est la performance : la recherche doit être rapide.

Ce genre de situation, moins courante, est à la base des modèles non-relationnels, ou NoSQL.

Mouvement NoSQL — *Définition*

« NoSQL désigne une famille de systèmes de gestion de base de données (SGBD) qui s'écarte du paradigme classique des bases relationnelles.

fr.wikipedia.org/wiki/NoSQL[1] »

Les bases de données NoSQL partagent des fondamentaux communs qui les opposent aux SGBDR :

- Identification par le système (*key-value*)
- Imbrication (*nested model*)
- Absence de schéma préalable (*schema-less*)
- Distribution des données (*sharding*)

Tous les SGBD NoSQL n'embrassent pas tous ces principes, mais tous en intègrent au moins un.

Types de SGBD NoSQL — *Exemple*

On distingue plusieurs types de bases de données NoSQL, deux grandes catégories émergent avec :

- Les base de données orientées **document** : elles gèrent des collections de documents arborescents, souvent au format JSON (exemples : MongoDB, OrientDB) ;
- Les bases de données orientées **graphe** : elles représentent l'information sous la forme de nœuds et d'arcs (exemples : Neo4J, OrientDB).

Signification de NoSQL — *Attention*

NoSQL signifie plutôt *Not Only SQL* que *No SQL*. Il s'agit en général de compléments aux SGBDR pour des besoins spécifiques et non de solutions de remplacement.

[1] https://fr.wikipedia.org/wiki/NoSQL

Exemple

Pour gérer un site de vente en ligne :

1. On utilisera une base relationnelle, comme PostgreSQL, pour enregistrer rigoureusement les transactions commerciales validées. Cela représente environ 1% de l'ensemble des accès aux données, les autres actions étant essentiellement de la consultation et de la construction de paniers d'achat.
2. On utilisera une base orientée **document**, comme MongoDB, pour gérer l'interaction entre l'utilisateur et le catalogue de produits et enregistrer les paniers. Cela permet de proposer une interface plus réactive. Certaines incohérences se manifesteront parfois, par exemple on aura autorisé à ajouter un objet au panier alors qu'au moment de la validation (et donc de la vérification dans la base PostgreSQL) on découvrira qu'il n'est finalement pas disponible. On considère ici que ces quelques erreurs sont non critiques et acceptables au regard du gain de performance acquis.

Identification par le système (clé-valeur)

Fondamental

Les bases NoSQL remettent en avant la fonction première des bases de données, **stocker et retrouver**.

Elles offrent une interface simple permettant de récupérer des données ou ensemble de données, souvent sous la forme d'un objet JSON, grâce à une clé artificielle.

Exemple

Avec MongoDB `db.Cart.find(ObjectId("5ecbc1d9d89c6e4a4ec10cdd"))` renverra le fichier JSON correspondant à la clé « *5ecbc1d9d89c6e4a4ec10cdd* ».

Identification universelles

Complément

Les clés artificielles utilisent des valeurs qui les identifient à l'échelle de la base de données, voire du monde :

- *Object Identifiers* (OID)
- *Universally Unique IDentifier* (UUID)
- *Uniform Resource Identifier* (URI)

Imbrication (JSON)

Fondamental

Les bases NoSQL permettent souvent de stocker des objets structurés (arbres, listes, etc.), souvent en JSON, ce qui permet de gérer des informations composées comme une seule donnée. Ceci conduit à des structures plus simples (et moins contrôlées).

Exemple

Avec MongoDB `db.Cart.find(ObjectId("5ecbc1d9d89c6e4a4ec10cdd"))` renverra :

```
{
  "_id" : ObjectId("5ecbc1d9d89c6e4a4ec10cdd")
  "login" : "stph",
  "mail" : "stphweb@czt.fr",
  "cart" : [
    {
    "code" : "A7142",
```

```
    "comnonName" : "Artichauts",
    "scientificName" : "Cynara cardunculus",
    "seed" : "Black star",
    "quantite" : 1,
    "price" : 3.10
    },
    {
    "code" : "A7142",
    "comnonName" : "Potiron",
    "scientificName" : "Cucurbita maxima",
    "seed" : "Orange planet",
    "quantite" : 2,
    "price" : 3.05
    }
  ]
}
```

Absence de schéma préalable (schema-less)

Fondamental

Les bases NoSQL se fondent sur une approche dite *« schema-less »*, c'est à dire sans schéma logique défini *a priori*.

L'équivalent du `CREATE TABLE` en SQL n'est soit pas nécessaire, soit même pas possible ; on peut directement faire l'équivalent de `INSERT INTO`.

Le but est d'assouplir la structure des données pour facilement pouvoir y accéder et faciliter la manipulation des données par les développeur.

Cela apporte de la souplesse et de la rapidité, mais se paye avec moins de contrôle et donc de cohérence des données : le traitement et la cohérence sont laissés à la couche applicative.

Distribution des données (sharding)

Complément

Le *sharding* est un mécanisme de partitionnement des données par enregistrement (ou partitionnement horizontal). Les enregistrements sont distribués sur des nœuds serveurs différents en fonction de la valeur de la clé.

Pour trouver à quelle machine est associée la clé de l'enregistrement, on utilise généralement des fonctions de hachage.

Le modèle de stockage clé-valeur facilite le stockage distribué entre plusieurs machines.

NoSQL et couche applicative

Complément

Étant donné que les bases de données NoSQL offrent moins de fonctions et renvoient des objets plus complexes, la couche applicative est plus conséquente qu'en SQL : il faut y transférer les fonctions qui ne sont plus assurées par le SGBD.

[cf. XqqpWBJX]

Exercice : Appliquer la notion

On dispose de données représentées sous une forme *schema-less* et imbriquées, en utilisant la formalisme JSON. Il représente des cours et des étudiants qui suivent ces cours.

```
{
  "code": "nos01",
  "title": "Introduction au NoSQL",
  "credits": 4,
  "students": [{
      "name": "Baggins",
      "firstname": "Bilbo"
    },
    {
      "name": "Took",
      "firstname": "Peregrin"
    }
  ]
}
```

Question

Proposer un code SQL permettant de représenter ces données dans une base de données relationnelle. On donnera les instructions `CREATE` et `INSERT` correspondant à ces données.

JavaScript Object Notation

[cf. 4Jy7Iwk6]

Mise en situation

Les bases de données relationnelles utilisent toutes la notion de table, ou tableau, pour représenter et stocker les données : table des personnes, table des locations, table des ingrédients, etc.

Dans les modèles non-relationnels, cette notion de table disparaît complètement. Comment sont représentées les données ?

Très souvent grâce au format JSON, qui est utilisé comme format d'entrée-sortie par de nombreuses bases de données NoSQL, du fait de sa simplicité. Il permet de structurer l'information en objets, arbres et tableaux.

Introduction

Fondamental

De nombreuses bases de données NoSQL utilisent JSON comme format de représentation des données.

JSON est un format de représentation logique de données, hérité de la syntaxe de création d'objets en JavaScript.

Un fichier JSON simple

Exemple

```json
{
  "name": "Norris",
  "givenname": "Chuck",
  "age": 73,
  "state": "Oklahoma"
}
```

JSON est indépendant de tout langage

Attention

Bien que JSON puise sa syntaxe du JavaScript, il est **indépendant de tout langage de programmation**. Il peut être interprété par n'importe quel langage. La plupart proposent aujourd'hui des *parsers* qui rendent cette opération simple.

Éléments du format JSON

Fondamental

Il existe deux types d'éléments :

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

Règles syntaxiques

§ *Syntaxe*

- Tout **fichier JSON bien formé** doit être :
  - **Soit un objet** commençant par « `{` » et se terminant par « `}` »,
  - **Soit un tableau** commençant par « `[` » et terminant par « `]` ».
- Un objet contient une liste de **paires clé-valeur** séparées par des **virgules**.
- Un tableau contient une liste de **valeur** séparées pas des **virgules**.

Valeurs possibles

Une valeur peut être d'un type primitif :

- Nombres : `int` et `float`
- Chaîne : `""`
- Booléen (`true` ou `false`)
- Absence de valeur : `null`

Une valeur peut être une structure :

- Tableau : liste de valeurs entre crochets, séparées par des virgules.
- Objet : listes de couples clé-valeur entre accolades, séparés par des virgules.

Tableaux et objets aussi autorisés dans les tableaux et objets (structure récursive).

? *Exemple*

```
{
  "id_cours": 1337,
  "nom_cours" : "IngDoc",
  "theme" : "Ingénierie 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"
    }
  ]
}

```

On a ici un objet avec quatre couples clé-valeur :

- L'entier « `id_cours` » ;
- La chaîne « `nom_cours` » ;
- La chaîne « `theme` » ;
- Le tableau d'objets « `etudiants` ».

Méthode

Il est nécessaire de se doter d'un outil de validation du JSON. Les éditeurs de code et les IDE en disposent, et on trouve également de nombreux validateurs en ligne.

Par exemple : jsonlint.com[1]

Extension

Complément

Un fichier au format JSON a généralement pour extension `.json`.

[cf. paCo9XRs]

[1] https://jsonlint.com

Exercice : Appliquer la notion

On dispose des informations suivantes sur le contact d'une personne :

- Prénom : John
- Nom : Smith
- Adresse : 42 Unity Parc Street, Brightonham East Sussex England BF1 2NE
- Numéro de téléphone :
    - Professionnel : +44 030 2134 8193, +44 030 2134 8194, +44 030 2134 8195
    - Personnel : +44 020 1934 3152
- Adresses mail :
    - Professionnel : john.smith@company.uk
    - Personnel : john@smith.uk

Question 1

Écrire en anglais un fichier JSON qui contient ces informations.

Question 2

On se donne le fichier JSON mal formaté suivant :

```
{
   "prenom": 'Robert',
   "nom" : 'Dupont',
   "address": '42 avenue du Parc, 42103 Saint-Barthélémy',
   "telephone_numbers": {
      "professional" : ["+33 4 72 31 32 12", "+33 4 72 31 32 13", "+33 4 72 31 32 14"],
      "personnal": [+33 9 78 70 31 12]
    },

   "email_adresses": {
      "professional": "robert.dupont@entreprise.fr",
      "personnal": "robert@dupont.fr"
    }
}
```

Trouver l'erreur à l'aide d'un formateur en ligne.

Indice :

On pourra par exemple utiliser ce formateur : jsonformatter.curiousconcept.com/[1]

[1] https://jsonformatter.curiousconcept.com/

Imbrication et structures arborescentes

[cf. QmGO6eDd]

Mise en situation

Certaines situations nécessitent de stocker les données sous forme imbriquée, un peu comme des poupées russes : une donnée en contient une autre, qui en contient une autre, etc.

Imaginez par exemple la représentation d'une page web : chaque page contient des bloc, qui contiennent des paragraphes, qui contiennent des lignes de textes, qui peuvent contenir des liens, etc.

L'utilisation des bases de données relationnelles pour gérer de telles données pose des problèmes. D'une part, le nombre de table va exploser, et d'autre part, la récupération des données ne pourra se faire qu'au prix de nombreuses jointures.

Dans ce genre de cas, il est pertinent d'utiliser d'autres formes de représentation des données, spécialisées dans l'imbrication.

Imbrication

 *Fondamental*

La représentation d'un arbre profond en relationnel implique la création de nombreuses tables et donc la création d'un modèle complexe éloigné de la réalité modélisée. Cette situation peut poser des problèmes de performance.

Comparaison représentation XML versus Relationnel

 *Exemple*

```xml
<?xml version="1.0" encoding="UTF-8"?>
<module>
    <titre>Introduction au non-relationnel</titre>
    <publication>10 avril 2018</publication>
    <licence>CC BY-SA</licence>
    <activite>
        <titre>Cours</titre>
        <section>
            <titre>Au delà du relationnel</titre>
            <page>
                <titre>Problème de l'imbrication</titre>
                <entete>
                    <auteur>Stéphane Crozat</auteur>
                    <date>11 avril 2018</date>
                </entete>
                <introduction>
                    <paragraphe>Si on manipule des structures de données imbriquées dans une
                        application...</paragraphe>
                    <paragraphe>En effet la représentation d'un arbre profond... </paragraphe>
                </introduction>
                <exemple>
                    <xml>
                        <fichier>exemple01.xml</fichier>
```

```
                    </xml>
                </exemple>
            </page>
        </section>
    </activite>
</module>

```

```
Module (#id:integer, titre:varchar, publication:date, licence:varchar)
Activite (#id:integer, #titre:varchar, #module=>Module)
Section (#id:integer, #titre:varchar, #activite=>Activite)
Page (#id:integer, #titre:varchar, #Section=>Section)
Entete (#page=>Page, auteur:varchar, date:date)
Bloc (#id:int, page=>Page, type:{introduction|exemple})
Paragraphe (#id:int, bloc=>Bloc, texte:CLOB)
Xml (#id:int, bloc=>Bloc, fichier:varchar)
```

Ici, il faut réaliser des jointures entre toutes les tables pour obtenir une information complète. Ce sera une opérations compliquée et coûteuse.

À l'inverse le découpage est peu utile, car les éléments enfants de modules ne sont pas **partageables**. Ils ne seront ici jamais référencés par autre chose qu'un module.

Objet partageable *Définition*

Un objet est **partageable** s'il peut entretenir des associations avec plusieurs autres objets.

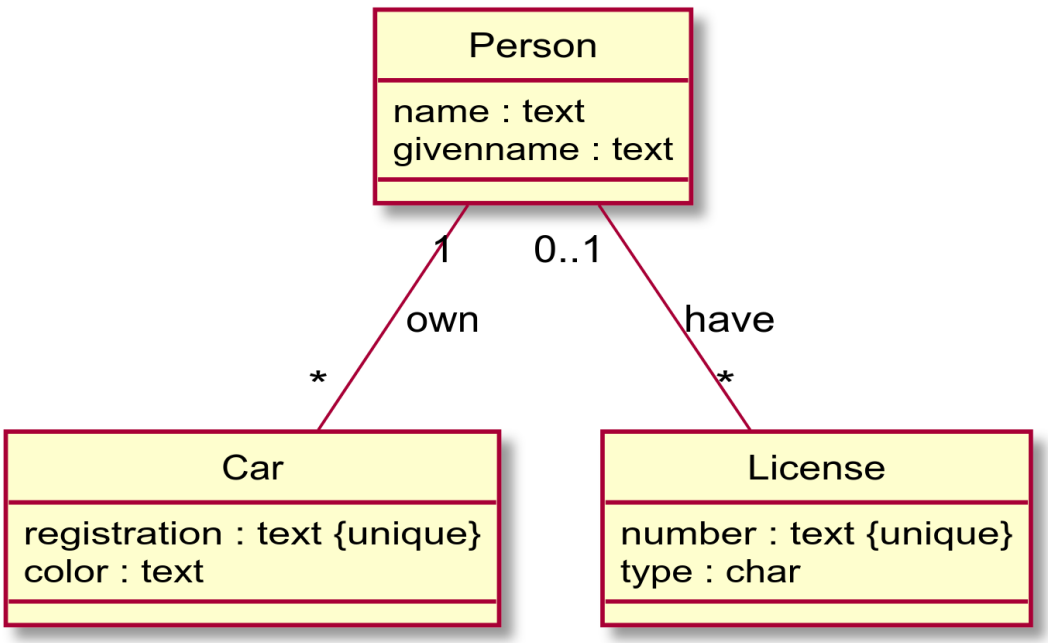
Objet non-partageable *Définition*

Un objet est **non partageable** s'il ne peut entretenir qu'au plus une seule association avec un seul autre objet.

En UML, cela signifie :

- Qu'il ne possède au plus qu'une seule association de type `1:N` ;
- Qu'il est du côté « *N* » de l'association.

Objets non-partageables *Exemple*



Ici :

- *Person* est partageable, entre plusieurs voitures et entre plusieurs permis.
- *Car* et *License* sont non-partageables. Une voiture ne peut que appartenir à une personne et un permis ne peut que être possédé par une personne.

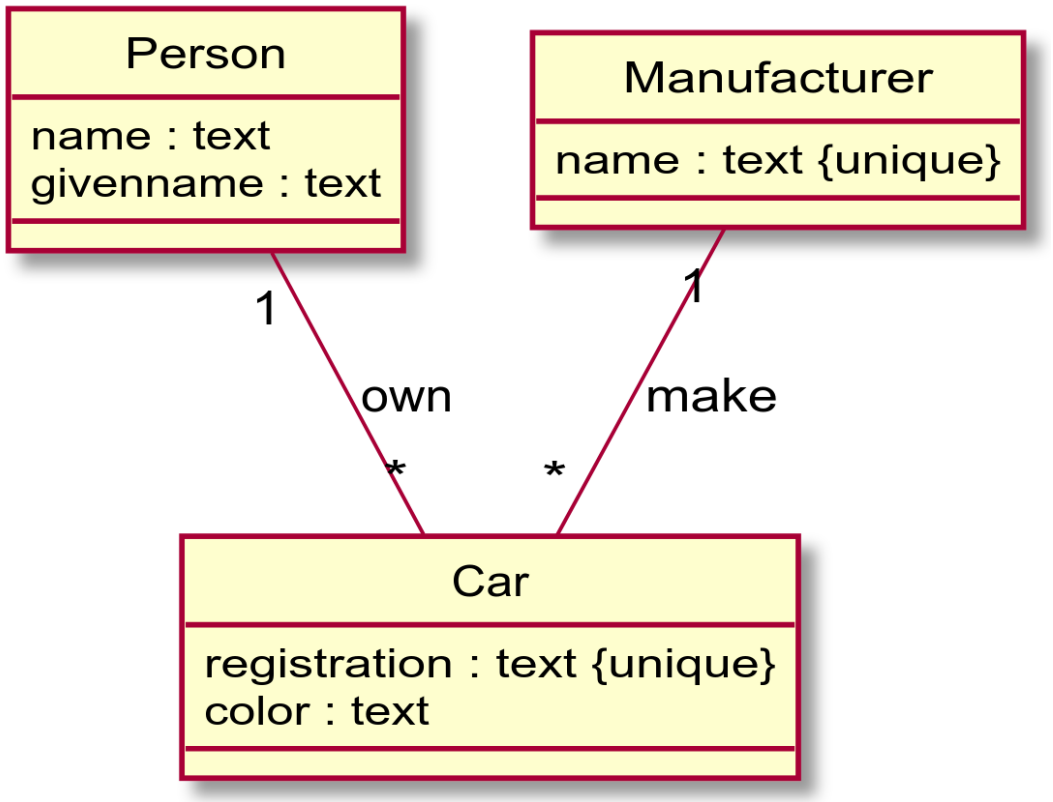
En conséquence, même si on regroupe toutes les informations dans un objet *Person*, aucune redondance n'est introduite.

Bien entendu, il faudra vérifier avec la couche applicative que des contraintes comme l'unicité du numéro d'immatriculation ou du numéro de permis sont respectées. On a perdu cette possibilité avec ce modèle.

```
{
  "person" : {
    "name" : "Stark",
    "givenname" : "Tony",
    "license" : [
      {
      "number" : "AX12854742",
      "type" : "B"
      }
    ],
    "car" : [
      {
      "registration" : "1999 CQ 60",
      "color" : "white"
      }
      ]
  }
}

```

Objet partageable ? *Exemple*



Ici *Car* est partagée entre *Person* et *Manufacturer*. Une représentation imbriquée conduirait systématiquement à la redondance : ici la répétition de *color*.

```
{
  "person" : {
    "name" : "Stark",
    "givenname" : "Tony",
    "car" : [
      {
      "registration" : "1999 CQ 60",
      "color" : "white"
      }
      ]
  }
}

```

```
{
  "manufacturer" : {
    "name" : "Citroën",
    "car" : [
      {
      "registration" : "1999 CQ 60",
      "color" : "white"
      }
      ]
  }
}
```

Attention

L'existence d'une association « *N:M* » dans un schéma rend l'utilisation de l'imbrication délicate : la redondance est alors certaine.

Types arborescents sous PostGreSQL

Complément

Certains SGBDR comme PostgreSQL ou Oracle proposent des types permettant l'imbrication. Cette approche permet de concilier les bénéfices du relationnel et de l'imbrication au sein d'un même système, lorsque les données à manipuler sont mixtes.

On pourra consulter les pages de la documentation PostgreSQL sur :

- Le type XML : https://docs.postgresql.fr/12/datatype-xml.html
- Le type JSON : https://docs.postgresql.fr/12/datatype-json.html

[cf. 0qp01gNN]

Exercice : Appliquer la notion

Soit la base de données suivante permettant de représenter des articles en général et illustrée avec l'article *As we may think* de Vannevar Bush publié en 1945.

```sql
CREATE TABLE article (
  id INTEGER PRIMARY KEY,
  title VARCHAR(255) UNIQUE NOT NULL,
  publication DATE,
  licence TEXT
);

CREATE TABLE page (
  number INTEGER,
  article INTEGER NOT NULL REFERENCES article(id),
  title VARCHAR(255) NOT NULL,
  PRIMARY KEY (number, article)
);

CREATE TABLE paragraph (
  id INTEGER PRIMARY KEY,
  page INTEGER NOT NULL,
  article INTEGER NOT NULL,
  content TEXT,
  FOREIGN KEY (page, article) REFERENCES page(number, article)
);

INSERT INTO article VALUES (1, 'As we may think', '1945-07-01', 'Copyheart');
INSERT INTO page VALUES (1, 1, 'Introduction');
INSERT INTO page VALUES (2, 1, 'Of what lasting benefit has been man''s use of science');
INSERT INTO paragraph VALUES (1, 1, 1, 'This has not been a scientist''s war; it has been a war in which all have had a part');
INSERT INTO paragraph VALUES (2, 1, 1, 'For the biologists, and particularly for the medical scientists, there can be little indecision');

```

Le relationnel est mal adapté à la représentation de document, comme en témoigne le modèle assez complexe mis en œuvre ici.

Question

Proposer une représentation *schema-less* et imbriquée en JSON de cet article.

Identification et structures en graphe

[cf. XDSxC28S]

Mise en situation

Imaginez que vous deviez gérer les utilisateurs d'un réseau social, et plus particulièrement leurs relations : A est ami avec B, mais suit également C, a partagé le contenu de D et de E, etc.

Si vous choisissez le modèle relationnel, ce genre de relations entre les personnes se traduira par de nombreuses tables qui associent par exemple la personne suiveuse avec la personne suivie.

Mais la multiplication de ce genre de tables pose des problèmes de performances, particulièrement parce que la récupération des informations nécessite de nombreuses jointures. De plus, cette représentation n'est pas très instinctive.

Il existe d'autres structures, appelées graphes, qui résolvent ce problème particulier.

Exemple

Soit une application Web reposant sur un graphe d'utilisateurs. On souhaite afficher sur chaque page d'un utilisateur les utilisateurs qui lui sont connectés.

Si l'application repose sur une base de données relationnelle, elle devra à chaque accès à une page faire une requête au serveur pour chercher qui sont les utilisateurs connectés par le graphe (au degré 2).

1. J'accède à Alice : je fais une requête qui remonte les données relatives à Alice et aux utilisateurs qui lui sont liés, je trouve Bob et Charlie.
2. J'accède à Bob : je dois faire une nouvelle requête pour obtenir les données relatives aux utilisateurs liés à Bob (David et Esméralda).
3. J'accède à Charlie : je dois faire une nouvelle requête pour obtenir les données relatives aux utilisateurs liés à Charlie (Francesca).

```sql
SELECT *
FROM links l
JOIN users u
ON l.user2=u.id
WHERE l.user1='alice'
```

Fondamental

Les bases de données orientées graphes utilisent une autre méthode de représentation, qui associe à chaque donnée des **pointeurs** sur d'autres données.

Par exemple ici, chaque *utilisateur* dispose d'un pointeur sur chacun de ses *utilisateurs* liés. On pourra obtenir ces données beaucoup plus facilement.

Exemple

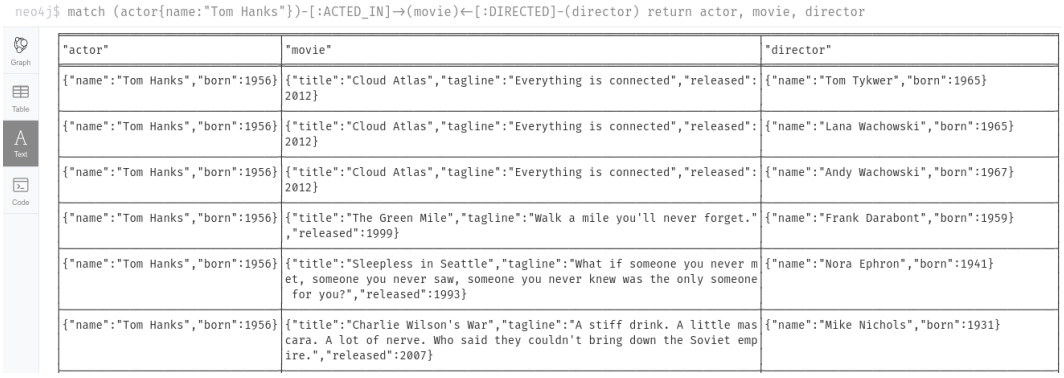
La requête suivante sous Neo4j permet de trouver tous les satellites qui sont des satellites du soleil.

```
1 MATCH (s1)-[:Satellite]-(s2)-[:Satellite]-(s3{name:"Sun"})
2 RETURN s1
```

Exemple

La requête suivante sous Neo4j permet de trouver tous les films dans lesquels a joué Tom Hanks, avec leur réalisateur.

```
1 MATCH (actor{name:"Tom Hanks"})-[:ACTED_IN]->(movie)<-[:DIRECTED]-(director)
2 RETURN actor, movie, director
```



neo4j$ match (actor{name:"Tom Hanks"})-[:ACTED_IN]→(movie)←[:DIRECTED]-(director) return actor, movie, director

"actor"	"movie"	"director"
{"name":"Tom Hanks","born":1956}	{"title":"Cloud Atlas","tagline":"Everything is connected","released":2012}	{"name":"Tom Tykwer","born":1965}
{"name":"Tom Hanks","born":1956}	{"title":"Cloud Atlas","tagline":"Everything is connected","released":2012}	{"name":"Lana Wachowski","born":1965}
{"name":"Tom Hanks","born":1956}	{"title":"Cloud Atlas","tagline":"Everything is connected","released":2012}	{"name":"Andy Wachowski","born":1967}
{"name":"Tom Hanks","born":1956}	{"title":"The Green Mile","tagline":"Walk a mile you'll never forget.","released":1999}	{"name":"Frank Darabont","born":1959}
{"name":"Tom Hanks","born":1956}	{"title":"Sleepless in Seattle","tagline":"What if someone you never met, someone you never saw, someone you never knew was the only someone for you?","released":1993}	{"name":"Nora Ephron","born":1941}
{"name":"Tom Hanks","born":1956}	{"title":"Charlie Wilson's War","tagline":"A stiff drink. A little mascara. A lot of nerve. Who said they couldn't bring down the Soviet empire.","released":2007}	{"name":"Mike Nichols","born":1931}


Mike Nichols
Robert Zemeckis
DIRECTED
DIRECTED
DIRECTED
The Polar Express
Charlie Wilson's War
That Thing You Do
Cast Away
ACTED_IN
ACTED_IN
ACTED_IN
DIRECTED
ACTED_IN
You've Got Mail
DIRECTED
Nora Ephron
DIRECTED
Ron Howard
DIRECTED
DIRECTED
Apollo 13
ACTED_IN
Tom Hanks
ACTED_IN
Sleepless
ACTED_IN
ACTED_IN
The Da Vinci Code
ACTED_IN
A League of Their Own
DIRECTED
Penny Marshall
ACTED_IN
ACTED_IN
ACTED_IN
Joe Versus the Volc...
The Green Mile
John Patrick Stanley
DIRECTED
Cloud Atlas
DIRECTED
Frank Darabont
DIRECTED
DIRECTED
DIRECTED
Lana Wacho...
Andy Wacho...
Tom Tykwer

Exercice : Appliquer la notion

On dispose d'un extrait d'un graphe d'un réseau social. Dans ce graphe, on représente 5 personnes : Agathe, Baptiste, Charline, David et Élise.

On dispose des liens *« suiveur-suivi »* suivants :

- Agathe suit Baptiste et Charline ;
- Baptiste suit Agathe, Charline et David ;
- Charline suit David ;
- David suit Baptiste et Charline ;
- Élise suit tout le monde mais n'est suivie par personne.

Question 1

Combien il y a-t-il de liens ?

Question 2

On souhaite représenter ces informations à l'aide d'un modèle relationnel. On met en place une table `liens` contenant les liens qui sont composés de deux attributs `suiveur` et `suivi` ayant pour valeur les prénoms des personnes dans les liens de réseau :

liens

suiveur	**suivi**
Agathe	Baptiste
…	…

Compléter ce tableau avec les données de liens.

Question 3

On souhaite connaître en parcourant la table `liens` les personnes suivant Baptiste. Combien de lignes du tableau faut-il parcourir ?

Question 4

On imagine maintenant que l'on a plus uniquement ces 5 personnes et leurs 12 liens mais 3212 personnes et 42531 liens.

On renseigne l'information de tous ces liens dans la table `liens`, combien de lignes du tableau faut-il parcourir pour connaître les personnes suivant Baptiste?

Cette représentation est-elle adaptée pour modéliser un réseau social ?

Essentiel

[cf. gc1JAw3P]

Les modèles non-relationnels, regroupés sous l'appellation NoSQL, s'affranchissent de la structure en tableau plat du modèle relationnel et de la plupart de ses contraintes.

Si la cohérence est impactée, la performance est en revanche améliorée pour certains cas spécifiques.

Par exemple, quand les données sont fortement imbriquées, ou quand elles forment naturellement un graphe, il peut être pertinent d'utiliser une base non-relationnelle pour les stocker et les traiter.

In fine, le choix dépend des besoins de l'application, et un compromis entre cohérence et performance peut être trouvé en utilisant les deux modèles simultanément : le relationnel pour stocker les données critiques, et le non-relationnel pour le reste.

Auto-évaluation

1. Exercice final

Exercice 1

Associer les principes relationnels suivants avec leur définition.

Il est nécessaire d'exprimer les règles de cohérence concernant les données *a priori*.

Les informations doivent être découpées afin que chaque case d'une relation ne contienne qu'une seule donnée la fois.

Toute relation contient au moins une clé.

Toute information n'est inscrite qu'une seule fois dans la base de données.

Identification par les données	Atomicité	Schéma	Non-redondance

Exercice 2

Associer les principes non-relationnels suivants avec les mots-clés correspondant.

JSON UUID INSERT

Identification par le système	Imbrication	*schema-less*

Exercice 3 : Quiz - Culture

Exercice

Les bases de données NoSQL sont une évolution des bases SQL ; elles les remplacent petit à petit, et on n'utilise plus les bases relationnelles dans les nouveaux projets informatiques.

- ○ Vrai
- ○ Faux

Exercice

Pour la gestion des transactions bancaires il est préférable d'utiliser une base de données...

- ○ Non relationnelle, pour payer ses achats rapidement.
- ○ Relationnelle, pour conserver l'intégrité des données bancaires.

Exercice

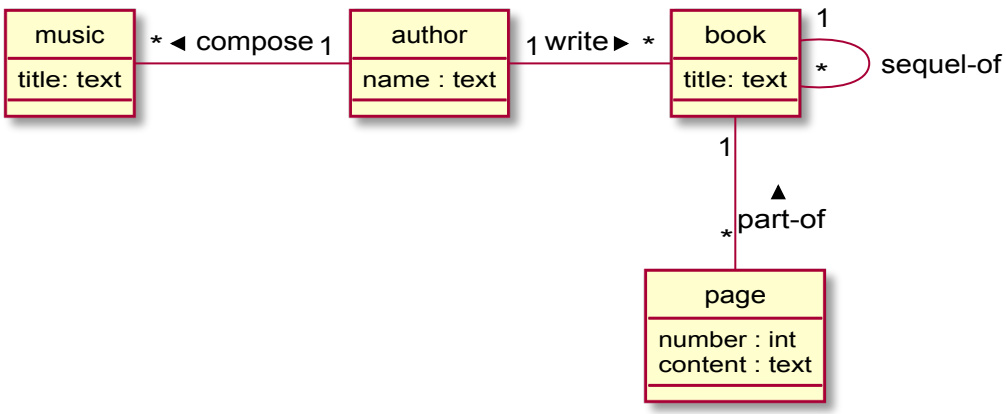
Dans une base de données relationnelle bien conçue, les informations présentes dans la base sont :

- ☐ Non redondantes
- ☐ Non nulles
- ☐ Atomiques

Exercice 7 : Quiz - Méthode

Exercice

Parmi les objets suivants, lesquels sont non-partageables et donc pourront être imbriqués sans introduire de redondance ?



- ☐ `page`
- ☐ `author`
- ☐ `book`
- ☐ `music`

Exercice

Parmi les déclarations suivantes, laquelle est correcte ?

- ```
  {
  "firstName": "Louis",
  "name": "Le Grand",
  "adress": ["Versailles", "Fontainebleau", "Chantilly"]
  }
  ```
- ```
  {
  firstName: "Louis",
  name: "Le Grand",
  adress: ["Versailles", "Fontainebleau", "Chantilly"]
  }
  ```
- ```
  {
  "firstName": "Louis",
  "name": "Le Grand",
  "adress": {"Versailles", "Fontainebleau", "Chantilly"}
  }
  ```

Exercice

Pour réaliser une base de données relationnelle contenant un tableau de voitures, quelle est la meilleure information pour différencier chaque voiture ? Cette donnée permettra d'identifier un enregistrement de la table.

- La date d'achat de la voiture
- La date d'immatriculation de la voiture
- Le nom du propriétaire de la voiture
- L'immatriculation de la voiture

Exercice 11 : Quiz - JSON

Exercice

Qu'est-ce que le JSON ?

- Un langage de programmation orienté objet
- Un type de pages web
- Un format qui permet de décrire des données structurées
- Une catégorie de documents générés par un traitement de texte

Exercice

Un fichier JSON doit forcément être traité via du code JavaScript.

- Vrai
- Faux
  ```

Exercice

On peut utiliser des tableaux en JSON et en XML.

- ○ Vrai
- ○ Faux, on ne peut le faire qu'en XML
- ○ Faux, on ne peut le faire qu'en JSON
- ○ Faux, on ne peut le faire ni en XML ni en JSON

Exercice 15 : Quiz - NoSQL

En parcourant la page Wikipédia du mouvement NoSQL : https://fr.wikipedia.org/wiki/NoSQL répondre aux questions suivantes :

Exercice

Quel est l'acronyme d'une notion, *qui n'est pas basique*, associée aux bases de données relationnelles et qui n'est pas respectée en général par les bases de données du mouvement NoSQL ?

- ○ ACID
- ○ SQL
- ○ UnQL

Exercice

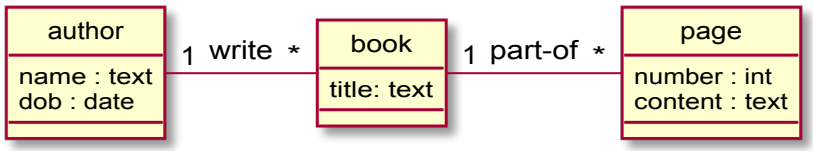
Les bases de données NoSQL sont en général plus adaptées à une conception reposant sur une architecture distribuée :

- ○ Vrai
- ○ Faux

Exercice 18 : Quiz - Code

Exercice

Quel est le code JSON *a priori* le plus adapté à ce modèle UML ?



- ☐ {
  "book": {
  "title": "La horde du contrevent",
  "author": {"name": "Damasio", "dob": "1969-08-01"},
  "pages": [{"number": 1, "content": "Nous sommes faits de l'étoffe dont sont tissés les vents."}, {"number": 2, "content": "À la cinquième salve, l'onde de choc fractura le fémur d'enceinte et ..."}]
  }}
- ☐ {
  "author": {
  "name": "Damasio",
  "dob": "1969-08-01",

"books": [
{"title":"La horde du contrevent",
"page": [{"number": 1, "content": "Nous sommes faits de l'étoffe dont sont tissés les vents."},
{"number": 2, "content": "À la cinquième salve, l'onde de choc fractura le fémur d'enceinte et ..."}]}
]
}}

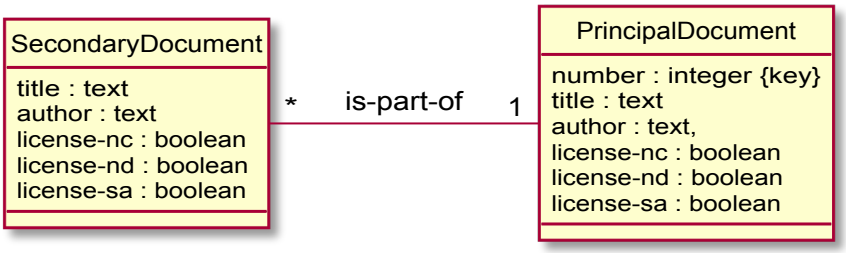
2. Exercice : Défi : Document sous licence Creative Commons

On souhaite créer une base de données permettant de d'assembler des documents situés dans une base de données. On envisage d'expérimenter une base de données NoSQL MongoDB, sachant que cette technologie est basée sur JSON.

Un document est décrit par un titre, un auteur, un nombre de pages et une licence *Creative Commons* (CC) : CC BY / CC BY-NC / CC BY-ND / CC BY-SA / CC BY-NC-ND / CC BY-NC-SA.

Un document peut être :

- Soit un document principal, il dispose alors toujours d'un numéro d'enregistrement unique dans la base, et son titre sera toujours renseigné ;
- Soit un document secondaire, il est alors membre d'un document principal.



Al Ringtu a écrit le document principal numéro `ec5c4388-b247-11ea-9db6-47c073ea731d` qui a pour titre *« Le grand livre des bases de données »*. Ce document a 42 pages. Il est sous licence CC BY-NC-SA.

Le document dispose des documents secondaires *« Le petit livre de l'UML »* de 12 pages et *« Le petit livre du SQL »* écrit par Jean de Nouvelhomme.

Question 1

Proposer une représentation JSON des données permettant de représenter le document principal seulement, sans les secondaires.

Question 2

Ajouter les documents secondaires au fichier JSON.

Question 3

Sachant que pour insérer un objet JSON dans une base MongoDB on utilise l'instruction : `db.NomDeLaCollection.insert(Objet)` (où *NomDeLaCollection* correspond à une table et *Objet* correspond à un enregistrement), insérer le document dans la collection `Principal`.

Question 4

Compléter le code MongoDB ci-dessous afin de trouver les documents écrits par Al Ringtu.

```
1 db.____________.find({"auteur" : ____________})
```

Question 5

Complémenter le code MongoDB ci-dessous afin de trouver les titres des documents secondaires des documents écrits par Al Ringtu.

```
1 db.Document.find({"author" : "Al Ringtu"}, {"____________" : 1})
```

Bibliographie

Bruchez Rudi. 2015. *Les bases de données NoSQL et le BigData : Comprendre et mettre en oeuvre*. 2ème édition, Eyrolles.

Espinasse Bernard. 2013. *Introduction aux systèmes NoSQL (Not Only SQL)*. http://www.lsis.org/espinasseb/Supports/BD/BD_NOSQL-4p.pdf.

w3resource. 2015. *NoSQL introduction*. http://www.w3resource.com/mongodb/nosql.php.

Index