Introduction aux bases de données non-relationnelles

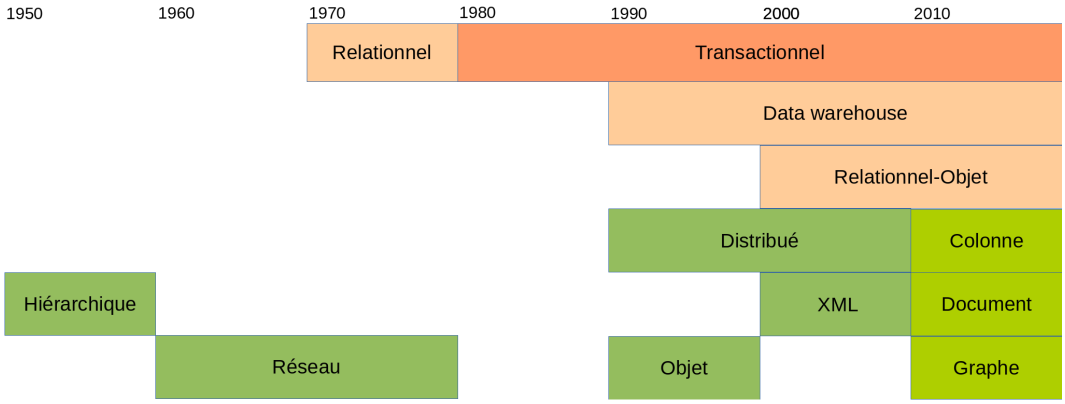
Table des matières

Cours

« Un couple de concepteurs de bases de données entre dans un restaurant NoSQL. Une heure après, ils ressortent sans avoir réussi à manger ; ils n'ont jamais réussi à trouver une seule table. »

1. Perspective technologique et historique : forces et faiblesses du relationnel

1.1. Relationnel et non-relationnel



Les BD NoSQL remettent en cause l'hégémonie des SGBDR telle qu'elle s'est constitué dans les années 1980.

Les BD NoSQL sont essentiellement un retour à des modèles de données antérieurs à cette hégémonie, telles que les représentations hiérarchique ou réseau qui existaient dans les années 60.

1.2. Domination du relationnel

Fondamental

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

Relationnel et contrôle de l'intégrité des données

À la naissance de l'informatique, plusieurs modèles de stockage de l'information sont explorés, comme les modèles hiérarchique ou réseau.

Mais c'est finalement le modèle relationnel qui l'emporte dans les années 1970 car c'est lui qui permet de mieux assurer le contrôle de l'intégrité des données, grâce à un modèle théorique puissant et simple.

On notera en particulier :

- Le **schéma** : on peut exprimer des règles de cohérence a priori et déléguer leur contrôle au système.
- La **normalisation** : on peut contrôler des contraintes (*via* le contrôle de la redondance) par un mécanisme de décomposition et retrouver l'information consolidée par les jointures.

- La **transaction** : le système assure le maintien d'états cohérents au sein d'environnements concurrents et susceptibles de pannes.

Relationnel et performance en contexte transactionnel

La représentation relationnelle se fonde sur la décomposition de l'information ce qui minimise les entrées/sorties (accès disques, transfert réseau) et permet d'être très performant pour répondre à des questions et des mises à jour ciblées (qui concernent peu de données parmi un ensemble qui peut être très grand). C'est donc une bonne solution dans un contexte transactionnel qui comprend de nombreux accès ciblés à la base.

En revanche ce n'est plus une bonne solution pour des accès globaux à la base (puisqu'il faut alors effectuer beaucoup de jointures pour reconsolider l'ensemble de l'information). C'est par exemple le problème posé notamment par le décisionnel.

2. Au delà des bases de données relationnelles : Data warehouse, XML et NoSQL

2.1. Problème de l'agrégat et des data warehouses

Problème posé par le décisionnel et résolu par les data warehouses *Fondamental*

- Décision vs Gestion
- Agrégat vs Accès ciblé
- Historisation vs Transaction

Définition

Un système décisionnel est une application informatique destinée à effectuer des exploitations statistiques sur la base des données existantes dans une organisation dans le but d'aider à la prise de décision.

- Le modèle relationnel est peu performant pour les agrégats qui portent sur de nombreuses tables car il est nécessaire de faire des jointures qui sont coûteuses.
- La rigueur du modèle relationnel n'est pas nécessaire pour des traitement statistiques qui sont tolérants aux incohérences isolées.
- Les volumes de données peuvent devenir importants si l'on conserve l'historique de toutes les transactions.

Agrégat *Exemple*

Une étude statistique peut rapidement concerner plusieurs dizaines de tables dans une base de données relationnelles et donc exiger autant de jointures.

Tolérance aux incohérences *Exemple*

Il n'est pas acceptable de perdre des données en contexte transactionnel (je ne sais pas si une personne existe ou pas), mais ce n'est pas important si je travaille sur une moyenne (l'âge moyen des personnes dans mon système ne sera pas impacté s'il me manque un enregistrement).

Volume de données *Exemple*

Si le système produit 1.000 enregistrements chaque jour et que je les conserve pendant 3 ans, j'ai 1.000.000 de lignes (mon système change d'ordre de grandeur).

- Un data warehouse est une base de données dédiée à un système décisionnel.
- Les problèmes d'agrégat, de tolérance aux incohérences et de volumes de données sont adressés par les data warehouses.
- Pour cela les data warehouses se basent sur des modèles **fortement redondants** et **potentiellement localement incohérents**.

Complément

Décisionnel *(cf. p.20)*

Différence entre un DW et un système transactionnel *(cf. p.20)*

Objectifs du modèle dimensionnel *(cf. p.21)*

2.2. Problème de l'imbrication et des structures arborescentes (XML ou Json)

Si on manipule des structures de données imbriquées dans une application, comme par exemple des structures arborescentes exprimées en XML ou en Json ou encore des documents HTML, il est difficile de stocker les informations dans une base de données relationnelle.

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), et pouvant 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 (#titre:varchar, publication:date, licence:varchar)
Activite (#titre:varchar, #module=>Module)
Section (#titre:varchar, #activite=>Activite)
Page (#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)
```

Impedance mismatch *Fondamental*

Par ailleurs les données étant manipulées en mémoire sous une certaine structure et en base sous une autre, il faut "jongler" entre plusieurs représentations et écrire du code de conversion ce qui est source de complexification et de risque d'erreur dans le processus de programmation.

2.3. Problème de l'identification et des structures en graphe

Si on manipule des données représentées en graphe, alors l'identification par les données implique de faire une requête pour chaque recherche dans le graphe, ce qui est potentiellement coûteux.

Exemple

Soit une application Web reposant sur un graphe d'utilisateurs et qui souhaite afficher sur chaque page consultée (une page concerne 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.

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

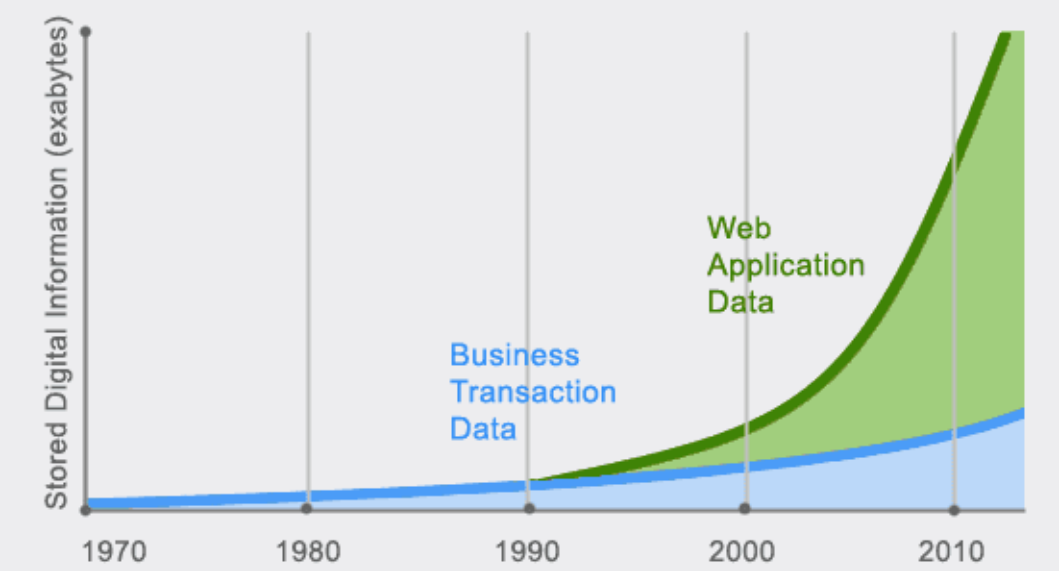
```
SELECT *
FROM links l JOIN users u
ON l.user2=u.id
WHERE l.user1='bob'
```

Fondamental

Si les données étaient identifiables selon une autre méthode, par exemple ici, si chaque *utilisateur* disposait d'un pointeur sur l'adresse physique de chacun de ses *utilisateurs* liés, alors on pourrait obtenir ces données beaucoup facilement.

2.4. Problème des gros volumes et de la distribution

Big data



Évolution des volumes de données d'entreprise versus web

Distribution *Fondamental*

Passage du serveur central (*main frame*) à des grappes de machines modestes :

- pour gérer l'explosion des données
- à cause de l'évolution du *hardware*

Problème de la performance

La performance de jointures exécutées sur plusieurs machines est mauvaise, or la normalisation augmente les jointures.

Problème de la gestion du contrôle d'intégrité et des transactions

il faut intervenir sur plusieurs machines en réseau à chaque modification des données et synchroniser les informations entre ces machines.

Problème de l'identification unique par les données

Comment savoir qu'une autre instance n'est pas en train d'affecter une même donnée clé.

Attention

Ces questions doivent être traitées sachant que les machines tombent en panne et que les réseaux sont victimes de coupures : est-ce que tout est mis en attente chaque fois qu'un élément du système dysfonctionne ?

- Si oui, plus le système est grand plus il est fragile, ce n'est pas tenable.
- Si non, on aura des incohérences, est-ce tenable ?

Fondamental

Il y a une équation à résoudre entre les paramètres de distribution, performance et cohérence.

2.5. Synthèse : De la 1NF à la NFNF (NF²)

1NF *Rappel*

La 1NF (*first normal form*) pose :

1. Les tables ont une clé.
2. Les attributs sont atomiques.

Identification : les tables ont une clé *Fondamental*

La première règle de la 1NF fixe le système d'identification des données en relationnel : ce sont les **données** qui permettent d'identifier les données.

C'est à dire que c'est un sous ensemble des données (une clé) qui permet d'identifier un enregistrement.

Or :

- Il existe d'autres solutions, par exemple l'identification par adresse mémoire (pointeur) ou disque, par URI, par UUID.
- Les clés artificielles sont déjà un échappatoire à cette règle puisqu'elles permettent l'identification avec une donnée spécifiquement créée pour l'identification, extérieure aux données.

Imbrication : les attributs sont atomiques. *Fondamental*

La seconde règle de la 1NF fixe le système de représentation des données en relationnel : la seule structure reconnue est la **case** d'un tableau.

C'est à dire qu'une case ne peut pas avoir de structure propre.

Or :

- Il existe d'autres structures de données : les arbres, les tableaux...
- Il existe déjà des types structuré ajouté au SQL depuis la version 1, comme la date introduite dès SQL2 en 1992.

Non First Normal Form (NFNF ou NF²) *Fondamental*

Les système en *Non First Normal Form* ou NF² sont des systèmes qui rompent volontairement avec la 1NF pour dépasser les limites qu'elle impose.

- Les attributs ne sont plus atomiques
- L'identification ne se fait plus par les données

Exemple

Les systèmes relationnels-objets et NoSQL sont des systèmes NF².

Stéphane Crozat (Contributions : Ulysse Meyer, Amadou Sonko, Harold Carrel-Billiard, Maxime Uzan, Cécile Fecherolle, les étudiants de API04 et de NF29 à l'UTC, 2013-2016)

3. Bases de données NoSQL

3.1. Définition du mouvement NoSQL

Définition

« Le NoSQL regroupe de nombreuses bases de données, récentes pour la plupart, qui se caractérisent par une logique de représentation de données non relationnelle et qui n'offrent donc pas une interface de requêtes en SQL.

http://blog.xebia.fr/2010/04/21/nosql-europe-tour-dhorizon-des-bases-de-donnees-nosql/ »

Attention

NoSQL signifie *Not Only SQL* et non pas *No SQL*, il s'agit de compléments aux SGBDR pour des besoins spécifiques et non de solutions de remplacement.

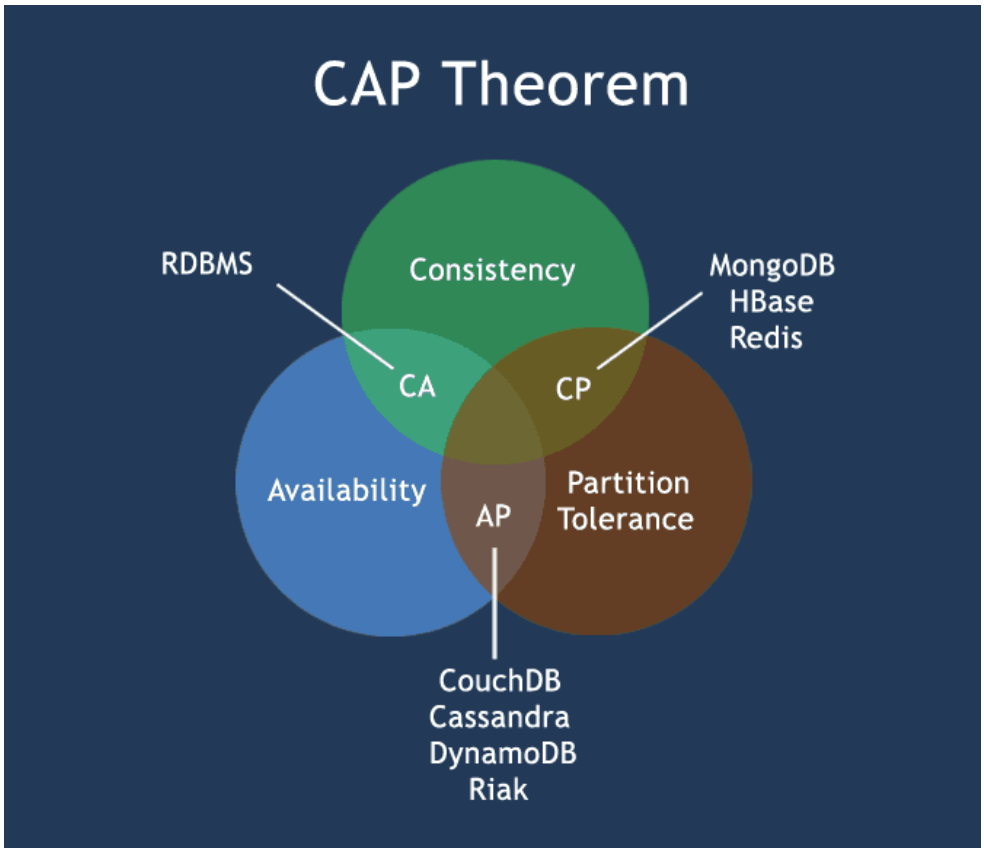
Exemple

- BD orientée clé-valeur
- BD orientée graphe
- BD orientée colonne
- BD orientée document

Complément

http://blog.xebia.fr/2010/04/21/nosql-europe-tour-dhorizon-des-bases-de-donnees-nosql/

3.2. "Théorème" CAP



Théorème CAP

Le "commerce" de la 3NF *Fondamental*

On échange de la performance contre de la souplesse ou contre de la cohérence.

3.3. Fondamentaux des modèles NoSQL : Clé-valeur, distribution, imbrication, schema-less

Fondamental

- Logique de dépôt uniquement (stocker et retrouver) ;
- c'est la couche applicative qui fait tout le travail (traitement, cohérence...).

Modèle de stockage clé-valeur

Les données sont des valeurs auxquelles on accède par une clé artificielle.

Les données ne sont plus des lignes stockées dans des tables.

Identification

Les clés d'identification sur des clés artificielles, en général unique à l'échelle de la BD, voire du monde :

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

Pointeurs physiques

On stocke des pointeurs physiques pour relier les données.

Imbrication

Structure des valeurs stockées connue par le serveur (RO, JSON, XML, structure interne de type colonne...)

Partitionnement et distribution (sharding)

Les données sont partitionnées horizontalement et distribuées sur les nœuds d'un cluster.

On utilise par exemple une clé de hashage pour distribuer les données.

Schema-less

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`.

Cela apporte de la souplesse et de la rapidité, mais se paye avec moins de contrôle et donc de cohérence des données.

Le mouvement NoSQL tend à réintégrer des fonctions de schématisation a priori, à l'instar de ce qui se fait en XML : le schéma est optionnel, mais conseillé en contexte de contrôle de cohérence.

3.4. Illustration des modèles NoSQL

Représentation de ventes en relationnel ? *Exemple*

Table Sales

#ticket	#date	#book
1	01/01/16	2212121504
1	01/01/16	2212141556
2	01/01/16	2212141556

Table Book

#isbn	#title	#author
2212121504	Scenari	1
2212141556	NoSQL	2

Table Author

#id	surname	firstname
1		
2	Bruchez	Rudi

Représentation de ventes en colonne

Exemple

Family Sales

#ticket	date	books
1	01/01/16	2212121504 2212141556
2	01/01/16	2212141556

Family Book

#isbn	title	a-surname	a-firstname
2212121504	Scenari		
2212141556	NoSQL	Bruchez	Rudi

Représentation de ventes en document

Exemple

Collection Sales

#oid	
4d0407660766b236450b45a3	"ticket" : 1 "date" : "01/01/16" "books" : ["_id" : 2212121504 "_id" : 2212141556]
4d0407660766b236450b45a4	"ticket" : 2 "date" : "01/01/16" "books" : ["_id" : 2212141556]

Collection Book

#oid	
4d0407660766b236450b45a5	"isbn" : 2212121504 "title" : "Scenari"

4d0407660766b236450b45a6	"isbn" : 2212141556
	"title" : "NoSQL"
	"author" : {
	"surname" : Bruchez
	"firstname" : Rudi
	}

Stéphane Crozat (Contributions : Ulysse Meyer, Amadou Sonko, Harold Carrel-Billiard, Maxime Uzan, Cécile Fecherolle, les étudiants de API04 et de NF29 à l'UTC, 2013-2016)

Représentation de ventes en document (avec imbrication redondante)

Exemple

Collection Sales

#oid	

4d0407660766b236450b45a3	"ticket" : 1
	"date" : "01/01/16"
	"books" : [
	{
	"isbn" : 2212121504
	"title" : "Scenari"
	}
	{
	"isbn" : 2212141556
	"title" : "NoSQL"
	"author" : {
	"surname" : Bruchez
	"firstname" : Rudi
	}
	}
	]

4d0407660766b236450b45a4	"ticket" : 2
	"date" : "01/01/16"
	"books" : [
	{
	"isbn" : 2212141556
	"title" : "NoSQL"
	"author" : {
	"surname" : Bruchez
	"firstname" : Rudi
	}
	}
	]

Représentation de ventes en graphe — *Exemple*

Classe Sales

#oid		
4d0407660766b236450b45a3	*property*	ticket : 1
	property	date : 01/01/16
	relation	book : 4d0407660766b236450b45a5
	relation	book : 4d0407660766b236450b45a6
4d0407660766b236450b45a4	*property*	ticket : 2
	property	date : 01/01/16
	relation	book : 4d0407660766b236450b45a6

Classe Book

#oid		
4d0407660766b236450b45a5	*property*	title : Scenari
4d0407660766b236450b45a6	*property*	title : NoSQL
	relation	author : 4d0407660766b236450b45a8

Classe Author

#oid		
4d0407660766b236450b45a8	*property*	surname : Bruchez
	property	firstnam : Rudi

4. Un exemple : Modélisation logique arborescente et objet en JSON

4.1. JavaScript Object Notation

Introduction — *Définition*

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

C'est un format réputé léger (il ne contient pas trop de caractères de structuration), assez facilement lisible par les humains, facilement *parsable* par les machines, et indépendant des langages qui l'utilisent (sa seule fonction est de décrire des données, qui sont ensuite utilisées différemment pour chaque cas suivant le contexte).

Un fichier JSON simple — *Exemple*

```
{
   "nom" : "Norris",
   "prenom" : "Chuck",
   "age" : 73,
   "etat" : "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 ainsi être interprété par tout langage à l'aide d'un *parser*.

Extension — *Complément*

Un fichier au format JSON a pour extension "**.json**".

4.2. 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"
                }
              ]
}
```

4.3. Principaux usages de JSON

Chargements asynchrones

Avec la montée en flèche des chargements asynchrones tels que l'AJAX (Asynchronous JavaScript And XML) dans le web actuel (qui ne provoquent pas le rechargement total de la page), il est devenu de plus en plus important de pouvoir charger des données organisées, de manière rapide et efficace.

Avec XML, le format JSON s'est montré adapté à ce type de besoins.

Les APIs

Des sociétés telles que Twitter, Facebook ou LinkedIn, offrent essentiellement des services basés sur l'échange d'informations, et font preuve d'un intérêt grandissant envers les moyens possibles pour distribuer ces données à des tiers.

Alors qu'il n'y a pas de domination totale d'un des deux formats (JSON ou XML) dans le domaine des APIs, on constate toutefois que JSON est en train de prendre le pas là où le format XML avait été pionnier.

APIs retournant des données au format JSON ? *Exemple*

Twitter : https://dev.twitter.com/rest/public : récupération de données du réseau social.

Netatmo : https://dev.netatmo.com/doc/publicapi : récupération de données météo

Les bases de données

Le JSON est très utilisé dans le domaine des bases de données NoSQL (MongoDB, CouchDB, Riak...).

On notera également qu'il est possible de soumettre des requêtes à des SGBDR et de récupérer une réponse en JSON.

Fonctions JSON de Postgres et MySQL ? *Exemple*

Fonctions PostgreSQL : http://www.postgresql.org/docs/9.3/static/functions-json.html

Fonctions MySQL : https://dev.mysql.com/doc/refman/5.7/en/json.html

4.4. Exercice

Exercice

Qu'est-ce que le JSON ?

- ○ Un langage de programmation orientée 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

A quel domaine le JSON n'est-il pas appliqué (à l'heure actuelle) ?

- ○ Le Big Data
- ○ Les chargements asynchrones
- ○ Les APIs
- ○ La mise en forme de pages web

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

4.5. Exercice : Un programme de traitement de JSON

Soit le fichier HTML et le fichier JavaScript ci-après.

```
1 <html>
2 <head>
3 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
4 <title>Démo JSON/JavaScript</title>
5 <script type="text/javascript" src="query.js"></script>
6 </head>
7 <body>
8 <input type="file" id="myfile" onchange="query()"/>
9 <div id="mydiv"/>
10 </body>
11 </html>
```

```
function query() {
// File
let vFile = document.getElementById("myfile").files[0];
// Reader
let vReader = new FileReader();
vReader.readAsText(vFile);
vReader.onload = function(pEvent) {
    // String Input
    let vContent = pEvent.target.result;
    // JSON to object
    let vJson = JSON.parse(vContent);
    // Query
    let vResult = vJson.prenom + " " + vJson.nom + " (" + vJson.age + ")";
    // Output
    document.getElementById("mydiv").appendChild(document.createTextNode(vResult));
};
}
```

Question 1

Écrire un fichier JSON permettant de représenter une personne avec les attributs suivants :

- un nom
- un prénom
- un age
- une liste d'adresses mail

Indice :

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

Question 2

Expliquer ce que fait le programme. Tester le programme avec le fichier JSON proposé.

Question 3

Compléter la fonction *query()* afin d'afficher la liste des adresses mail (en plus des information actuelles).

Indice :

On parcours le tableau JSON à l'aide d'une boucle.

```
for (let i=0;i<vJson.adresse.length;i++) {
  ...
}
```

Exercice

1. Exercice : Modélisation orientée document avec JSON

On souhaite réaliser une base de données orientée documents pour gérer des cours et des étudiants, étant données les informations suivantes :

- Un cours est décrit par les attributs code, titre, description, crédits et prérequis.
- Les prérequis sont d'autres cours.
- Un étudiant est décrit par les attributs nom, prénom, adresse.
- Les adresses sont composées d'un numéro de rue, d'une rue, d'une ville et d'un code postal.
- Un étudiant suit plusieurs cours et les cours sont suivis par plusieurs étudiants.

Question 1

Réaliser le MCD en UML de ce problème.

Question 2

Proposer un exemple JSON équivalent à ce que l'on aurait fait en relationnel normalisé (sachant que ce type de solution ne sera généralement pas favorisé en BD orientée document).

Indice :

Représentation d'un cours par un objet JSON.

```
{
"code":"api04",
"titre":"DW et NOSQL"
...
}
```

Question 3

Proposer un exemple JSON basé sur l'imbrication.

Quel est le principal défaut de cette solution ?

Est-il toujours possible d'avoir une solution ne reposant que sur l'imbrication ?

Question 4

Proposer un exemple JSON basé sur les références.

Quelles sont les principales différences avec un système relationnel ?

Question 5

Sachant que l'objectif de l'application est de visualiser une liste des étudiants avec les cours que chacun suit, et d'accéder aux détails des cours uniquement lorsque l'on clique sur son code ou son titre.

Proposer une solution adaptée à ce problème mobilisant référence et imbrication.

Contenus annexes

1. Décisionnel

Définition

« Le système d'information décisionnel est un ensemble de données organisées de façon spécifiques, facilement accessibles et appropriées à la prise de décision [...].

La finalité d'un système décisionnel est le pilotage d'entreprise.

Les systèmes de gestion sont dédiés **aux métiers** de l'entreprise [...].

Les systèmes décisionnels sont dédiés **au management** de l'entreprise [...].

(Goglin, 2001, pp21-22)∗ »

Synonymes : informatique décisionnelle, *business intelligence*, *BI*∗

2. Différence entre un DW et un système transactionnel

BD transactionnelle

Une base données classique est destinée à assumer des **transactions** en temps réel :

- Ajout, mise à jour suppression de données
- Questions sur des données identifiées ou questions statistiques

Datawarehouse

Un *DW*∗ est uniquement destiné à l'exécution de **questions statistiques** sur des données statiques (ou faiblement dynamiques).

PRIMITIVE DATA/OPERATIONAL DATA	DERIVED DATA/DSS DATA
• application oriented	• subject oriented
• detailed	• summarized, otherwise refined
• accurate, as of the moment of access	• represents values over time, snapshots
• serves the clerical community	• serves the managerial community
• can be updated	• is not updated
• run repetitively	• run heuristically
• requirements for processing understood a priori	• requirements for processing not understood *a priori*
• compatible with the SDLC	• completely different life cycle
• performance sensitive	• performance relaxed
• accessed a unit at a time	• accessed a set at a time
• transaction driven	• analysis driven
• control of update a major concern in terms of ownership	• control of update no issue
• high availability	• relaxed availability
• managed in its entirety	• managed by subsets
• nonredundancy	• redundancy is a fact of life
• static structure; variable contents	• flexible structure
• small amount of data used in a process	• large amount of data used in a process
• supports day-to-day operations	• supports managerial needs
• high probability of access	• low, modest probability of access

Un changement d'approche, extrait de (Inmon, 2002, p15)

3. Objectifs du modèle dimensionnel

La modélisation par schéma en étoile, par opposition aux schémas normalisés en 3NF, permet de répondre à deux besoins caractéristiques des systèmes décisionnels : la **performance** et la **simplicité** des requêtes.

Performance

En effet en tant que structures **redondantes** les schémas en étoiles permettent d'agréger la table des faits avec n'importe qu'elle dimension en **une seule opération de jointure** (deux ou trois pour les schémas en flocons).

Ce gain de performance est souvent critique puisque les volumes de données sont généralement d'un ordre de grandeur très supérieur à celui des systèmes transactionnels.

Cette redondance ne pose pas les mêmes problèmes que dans les systèmes transactionnels, en effet :

- les données étant statiques (importées), il n'y a pas de risque de divergence d'information lors de mises à jour
- l'usage du *datawarehouse* étant essentiellement statistique (regroupement), la conséquence d'une éventuelle erreur n'est pas du même ordre que dans un système transactionnel.

Simplicité

La présentation en étoile des données, avec les faits au centre et les dimensions autour, est particulièrement adaptée à **l'écriture rapide de requêtes simples** pour agréger des données de la table des faits selon des regroupements sur les tables de dimensions.

L'enjeu est de pouvoir répondre simplement et rapidement à une question simple, tandis qu'un modèle transactionnel, qui répond à d'autres contraintes, nécessitera souvent un code SQL complexe et des opérations multiples pour répondre à la même question. Cela permet notamment aux utilisateurs finaux de construire facilement de nouvelles requêtes au fil de leur exploration des données.

Caractéristiques d'un bon modèle décisionnel *Fondamental*

- Être performant pour le calcul d'agrégats sur de gros volumes de données (exploration de données, *reporting*)
- Être compréhensible par un utilisateur final, en particulier pour formuler facilement des requêtes (exploration de données)
- Être suffisamment performant au chargement pour répondre aux sollicitations de mise à jour (*ETL*∗)
- Être évolutif en fonction des évolutions amont (sources transactionnels) et aval (besoins d'exploitation) (ETL, métadonnées)

Abréviations

BI : Business Intelligence
DW : Data Warehouse
ETL : Extraction, Transformation, Loading

Bibliographie

Goglin J.-F. (2001, 1998). *La construction du datawarehouse : du datamart au dataweb*. Hermes, 2ème édition.

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

Crédits des ressources

Évolution des volumes de données d'entreprise versus web p. 7
http://creativecommons.org/licenses/by-nc-sa/3.0/fr/,
http://www.w3resource.com/mongodb/nosql.php

Théorème CAP p. 10
http://creativecommons.org/licenses/by-nc-sa/3.0/fr/,
http://www.w3resource.com/mongodb/nosql.php