# L3 - Bases de données non traditionnelles

Francois.Rioult@unicaen.fr

2 mars 2021

# Table des matières

# Chapitre 1

# JSON

## 1.1 Introduction

JSON est l'abréviation de JavaScript Object Notation.

C'est un format pour les données, une alternative à XML, qui offre l'immense avantage d'être nativement pris en charge par un langage : le *javascript*.

JSON permet de représenter des objets, contenant d'autres objets ou des variables, représentant les valeurs pour les champs. La grammaire du langage est la suivante : ($ws$ pour *white space*)

```
json
    element

elements
    element
    element ',' elements

element
    ws value ws

value
    object
    array
    string
    number
    "true"
    "false"
    "null"

object
    '{' ws '}'
    '{' members '}'
```

```
members
    member
    member ',' members

member
    ws string ws ':' element

array
    '[' ws ']'
    '[' elements ']'
```

Exemple :

```
{
    "items": [
        {
            "name": "index",
            "label": "Accueil"
        },
        {
            "name": "authors",
            "label": "Auteurs"
        },
        {
            "name": "conferences",
            "label": "Conférences"
        },
        {
            "name": "graph",
            "label": "Graph"
        }
    ]
}
```

## 1.2 En ligne de commande

En ligne de commande, on peut utiliser l'utilitaire `jq`, qui permet de définir des filtres sur des flux JSON, à la manière de `sed` :

```
$ cat names.json
[{"id": 1, "name": "Arthur", "age": "21"},{"id": 2, "name": "Richard", "age": "3
$ jq '.' names.json # pretty print : on demande la racine (.)
[
  {
    "id": 1,
```

```
    "name": "Arthur",
    "age": "21"
  },
  {
    "id": 2,
    "name": "Richard",
    "age": "32"
  }
]
$ cat names.json | jq '.[]' # le contenu du tableau
{
  "id": 1,
  "name": "Arthur",
  "age": "21"
}
{
  "id": 2,
  "name": "Richard",
  "age": "32"
}
$ cat names.json | jq '.[1]' # le premier élément du tableau
{
  "id": 2,
  "name": "Richard",
  "age": "32"
}
$ cat names.json | jq '.[].name'    # les champs name
"Arthur"
"Richard"
$ cat names.json | jq '.[1] | .name' # enchaînement de filtres
"Richard"
```

# Chapitre 2

# Modélisation des données NO-SQL

La finalité des données doit guider le processus de modélisation : qui va les utiliser, pourquoi, va-t-on les visualiser ? etc.

S'il ne faut pas mettre de côté les bonnes résolutions initiales des bases de données traditionnelles – les formes normales et les outils de modélisation comme UML, il est essentiel de concevoir le modèle de stockage au plus proche de celui de l'exploitation des données.

C'est le cas *emph* de SQL, dont données sont des tables, et les résultats de requêtes sont des tables.

En NO-SQL, le format des données est beaucoup plus riche, typiquement un arbre de données, exprimable par XML ou JSON. Il y a donc de multiples façons d'envisager la relation entre l'entrée et la sortie des données.

## 2.1 Exemple 1

On souhaite visualiser l'évolution au cours du temps des positions des joueurs d'une partie de DOTA. L'interface propose un assenceur temporel qui modifie la position des joueurs en intervenant sur les attributs HTML des éléments SVG générés par *map-reduce* sur l'*intégralité* des données : l'interface est générée par filtrage du flux de données.

La modélisation UML est à la figure 2.1. En normalisant de façon traditionelle, on aurait un paquet de relations et une requête de jointure monstreuse pour obtenir une sérialisation comme ci-dessous :

```
[
  {
    "id": 1,
    "team": 0,
    "data": [
```

```
{
  "x": 173,
  "y": 112,
  "tick": 1000
},
{
  "x": 173,
  "y": 112,
  "tick": 2000
},
{
  "x": 173,
  "y": 112,
  "tick": 3000
},
```


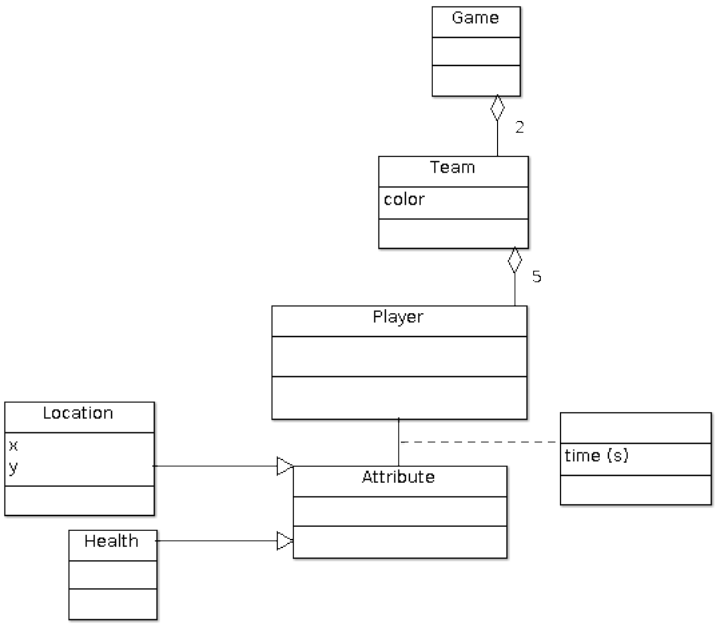


FIGURE 2.1 – Modèle des données de DOTA

## 2.2 Exemple 2

On veut développer un outil d'analyse des messages publiés sur Tweeter. L'utilisateur peut paramétrer une requête comme il le ferait sur Tweeter et le résultat est

enregistré en base de données (figure 2.2).

Les données peuvent être d'un volume considérable : plusieurs dizaines de milliers sur des mots-clés tendance ne sont pas à exclure.

Il serait tentant de stocker dans un unique document le résultat d'une requête, associée à ses paramètres, mais la taille de ce document risquerait d'être trop importante. Il est prudent de ne pas limiter le nombre de Tweets accessibles ; il ne faut pas les confiner dans un unique document.

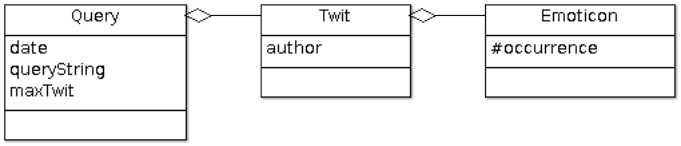


FIGURE 2.2 – Modèle des données pour TWITRACKER


```
{
        "_id" : ObjectId("5ad4c74028504c94fb67ad67"),
        "id" : "1523894065-14686",
        "date" : "1523894065",
        "username" : "@Manuimorou",
        "since" : "2018-01-01",
        "tweets" : [
                {
                        "username" : "Manuimorou",
                        "date" : "2018-04-16 08:08",
                        "replies" : 0,
                        "retweets" : 0,
                        "likes" : 23,
                        "geo" : "",
                        "mentions" : "",
                        "hashtags" : "",
                        "id" : "985792066973249536",
                        "permalink" : "https://twitter.com/Manuimorou/status/985
                        "emoticons" : 1,
                        "emoticonsStr" : "Homme haussant les epaules (teint de p
                        "replyings" : 1,
                        "replyingTo" : "Decathlon",
                        "text" : "Du foot"
                },
                ...
        ]
        }
...
```

]

# Chapitre 3

# Node.js

Node.js permet de programmer aisément des serveurs en utilisant JavaScript.

C'est une technologie qui est appréciée pour ses performances : le code applicatif n'est pas interprété après traitement par Apache comme dans le cas de PHP, mais il réside en mémoire.

Modèle client server, ex. PHP :

— un serveur distribue les fichiers et renvoie les résultats d'exécution des scripts

Modèle Node.js :

— on programme le code d'un serveur selon deux aspects :
  — la requête HTTP qui est effectuée auprès du serveur node, qui obéit au modèle client serveur
  — les opérations sur les socket, qui permettent au serveur de travailler en mode "push" : le client et le serveur se programment de la même façon.

Node est un écosystème à part entière, géré par le *node packet manager*, qui permet d'installer des modules avec la commande :

```
sudo apt-get install npm
npm install assert d3 mongo...
```

La contrepartie à cette simplicité est que les problèmes de fiabilité sont répartis sur les infrastructures, qui doivent gérer les serveurs node.

## 3.1 Architectures d'une application web

## 3.2 Serveur de fichier

Le code ci-dessous présente la réalisation d'un serveur minimal en node.

Les premières lignes décrivent les imports nécessaires.

La création du serveur HTTP indique le traitement à appliquer lors de la réception d'une requête. Ici, on décode l'adresse de la page demandée et les paramètres de la requête.

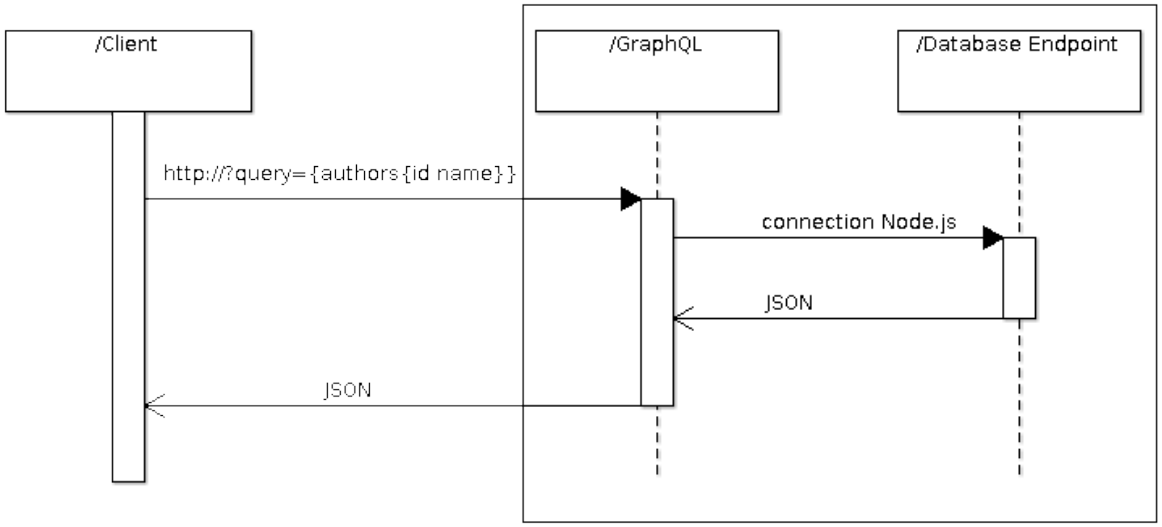


FIGURE 3.1 – Séquence d'une application avec Node/GraphQL

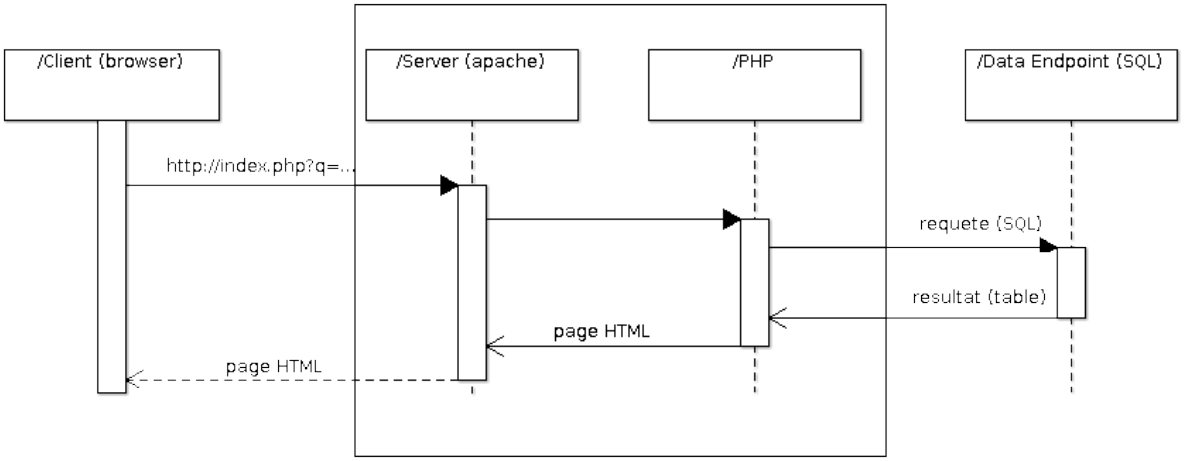


FIGURE 3.2 – Séquence d'une application avec PHP

Ensuite, on lit le fichier dans le répertoire adéquat.
Enfin, on envoie la réponse.

```
var http = require('http');
var url = require('url');
var querystring = require('querystring');
var fs = require('fs');

var server = http.createServer(function(req, res) {
    var page = url.parse(req.url).pathname;
    var params = querystring.parse(url.parse(req.url).query);

    fs.readFile(__dirname + '/' + page,
            function(err, data) {
                if (err) {
                    res.writeHead(500);
                    return res.end('Error loading ' + page);
                }
                console.log('sending page ' + page);
                res.end(data);
            });

});

var port = 8080;
console.log("listening to " + port);
server.listen(8080);
```

## 3.3 Serveur de fichier + socket

Ajoutons maintenant la capacité au client d'interagir avec le serveur. L'objet de la communication utilise principalement le JSON. C'est la combinaison de :

— un identifiant pour le message
— un contenu, souvent du JSON, a priori toute forme sérialisable

Par exemple :

```
server.emit('id', {player: -1, referee: -1});
```

Le client peut ainsi en définir des actions à déclencher en cas de réception d'un message :

```
<!DOCTYPE html>
<html>
    <head>
       <script src="/socket.io/socket.io.js"></script>
    </head>
    <body>
```

```
<header>
    <menu>
        <div id="message"></div>
    </menu>
</header>
<script>
    //-----------------------------------------
    // adresse regulierement ecoutee
    var serverUrl = "http://127.0.0.1";
    var serverPort = 8080;
    var server = io.connect(serverUrl + ":" + serverPort);

    server.on('message', function(data) {
        document.getElementById('message').innerHTML += 'received from th
    });

    server.emit('ready');

</script>
```

Côté serveur, c'est le framework *Express* qui est utilisé, qui améliore HTTP avec la notion de routes. Noter le peu de modification avec la version HTTP pour la partie serveur de fichier.

```
var fs = require('fs');
var url = require('url');
var querystring = require('querystring');

// npm install basex socket.io express socket.io-client
var express = require('express');
var app = express();

app.get('/*', function(req, res) {
    var page = url.parse(req.url).pathname;
    var params = querystring.parse(url.parse(req.url).query);

    fs.readFile(__dirname + '/html/' + page,
            function(err, data) {
                if (err) {
                    res.writeHead(500);
                    return res.end('Error loading ' + page);
                }
                console.log('sending page ' + page);
                res.end(data);
            });

    // sending response OK
```

```
});

var port = 8080;
console.log("listening to " + port);
var io = require('socket.io').listen(app.listen(port), {log: true});

// when the client is ready
io.sockets.on('connection', function(socket) {
    socket.on('ready', function(data) {
        console.log('received', 'ack');
        socket.emit("message", "welcome !");
    });
});
```

## 3.4 API standards

Node.js fournit des API standards :

— File : fichiers
— CLI : commande en ligne, pour faire des applications dans le terminal
— TCP : data byte
— HTTP : XHTML, formulaires

En outre, tout bon gestionnaire de base de données dispose d'un connecteur utilisable par Node, qui recoît alors du JSON.

## 3.5 API Web

— Express (infrastructure d'applications, fournit entre autres des routes et la gestion de middleware).
— Socket.IO (pour WebSocket et développer des application *push*, avec notification (contact du client sans sollicitation).

## 3.6 Difficulté à installer node/npm ?

On peut utiliser `nvm`, le gestionnaire de versions de `node` : `https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-18-04-fr`

Conflit de paquet avec node-gyp :

```
sudo apt install libssl1.0-dev
sudo apt install nodejs-dev
sudo apt install node-gyp
sudo apt install npm
```

Sinon, essayer avec aptitude.

# Chapitre 4

# Introduction à NoSQL - MongoDB

Pour Not Only SQL, ces systèmes simplifient la structure du système d'information en limitant au système clé - valeur. On evite les lourdeurs de SQL au profit d'une simplicité dans l'ajout d'information et son accès. En NoSQL, les contraintes d'intégrité sont plus faibles que dans les systèmes traditionnels SQL, mais ces contraintes peuvent être assouplies dans le cas de systèmes majoritairement fondés sur la lecture.

Les système NoSQL interviennent dans le contexte *Big Data* des acteurs majeurs du web, qui ouvrent leurs solutions : Cassandra par Twitter et Digg, MongoDB (Sourceforge).

Intérêt par rapport aux systèmes traditionnels SQL :

— la répartition des calculs et des données est nativement prise en compte
— bien adapté pour gérer des documents structurés compatibles avec JSON
— langage évolué pour traiter les requêtes, qui deviennent des programmes complets pour manipuler les données
— souplesse de création dans les collections, format *libre* de document

Inconvénients :

— la rigueur de conception des systèmes classiques peut faire défaut : on peut rapidement faire n'importe quoi
— la structure des données doit être équivalente en entrée et en sortie du système. Entre autres, il n'est pas prévu de faire des jointures et plus généralement, la manipulation de plusieurs collections au cours d'une même requête est vivement déconseillée. Si vraiment on ne peut pas s'en passer, revenir au SQL traditionnel.
— les calculs sont *faciles* à définir, *lents* à exécuter. Il faudra parfois alourdir les données avec des index longs à calculer.

Il est fondamental de comprendre que le système ne pourra travailler que sur une collection, *a priori* sans pouvoir accéder à une autre collection (sous peine de faire au pire un produit cartésien, au mieux une jointure). Ce travail consistera à appliquer des traitements sur les collections avec le framework *map-reduce* : une fonction (ou

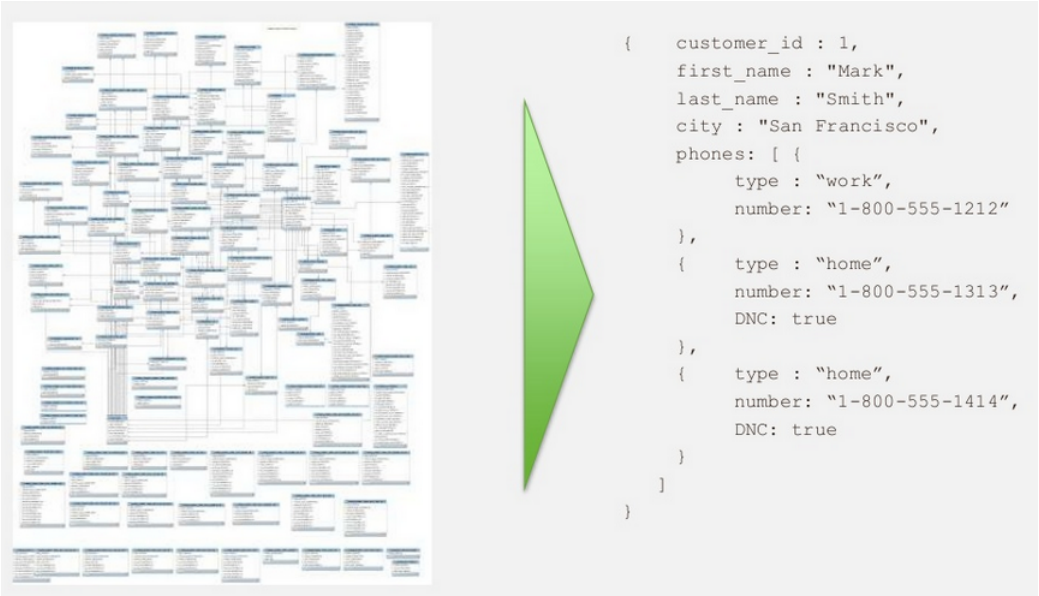


FIGURE 4.1 – `https://www.mongodb.com/blog/post/mongodb-vs-sql-day-1-2`

plusieurs à la suite l'une de l'autre) sont appliquées pour transformer les items ; on peut terminer le traitement en réduisant le résultat à un agrégat.

## 4.1 MongoDB

MongoDB gère des données richement structurées, maps de maps de liste sur des types de base : entier, flottant, date, chaînes. Il y a une symétrie entre les données et le schéma, selon que la donnée entre ou sorte.

Le système est organisé en *database* qui contiennent des *collections* de documents. Chaque document a un identifiant noté `_id` utilisé comme clé primaire pour optimiser la recherche (hachage). S'il n'est pas précisé par l'utilisateur, il est généré par le système.

On ajoute les données sous forme de BSON, un format JSON étendu pour gérer les types, dont les clés peuvent comporter un '$'. On dispose d'un *vrai* langage de requête (JavaScript [1]). Accès favorisé : les curseurs, une liste renvoyée par la requête, sur laquelle on va itérer une fonction.

La méthode *save()* insère ou met à jour si l'élement existe.

1. most interactions with MongoDB do not use JavaScript but use an idiomatic driver in the language of the interacting application.

### 4.1.1 Requetage

La requête se fait en deux temps :

1. critère : on indique un patron pour les documents sélectionnés

```
{ qty: { $gt: 25 } }
field: { $gt: value1, $lt: value2 }
{ _id: { $in: [ 5,  ObjectId("507c35dd8fada716c89d0013") ] } }
awards: {
          $elemMatch: {
               award: "Turing Award",
               year: { $gt: 1980 }
          }
}
name: {
         first: "Yukihiro",
         last: "Matsumoto"
       }
}
{
     "name.first": "Yukihiro",        // !!!!!! guillemets
     "name.last": "Matsumoto"
}
```

2. projection :

```
{ field1: <boolean>, field2: <boolean> ... }
{
  _id: 0,
  'name.last': 1,
   contribs: { $slice: 2 }         // les deux premiers éléments
}
```

Exemple :

```
db.ti5.find({game_winner:2}, {_id:0,match_id:1})
db.ti5.find({},{"game_info.dota.player_infos.player_name":1,
"game_info.dota.player_infos.game_team":1, "*.radiant_team_id":1})
```

### 4.1.2 Aggrégation

Une solution simple consiste à utiliser les groupes :

```
db.records.group( {
   key: { a: 1 },
   cond: { a: { $lt: 3 } },
   reduce: function(cur, result) { result.count += cur.count },
   initial: { count: 0 }
} )
```

— key : le champ sur lequel porte l'aggrégation
— cond : une condition sur les éléments sélectionnés
— reduce : une fonction appelée pour chaque enregistrement avec en paramètre l'élément courant et le résultat à produire
— initial : initialisation du résultat
La commande *group* produit la liste des clés des documents respectant la condition, leur applique une opération pour calculer un résultat aggrégé.

**Détails techniques**

L'installation est simple avec un paquet dédié :

```
apt-get install mongodb
```

On démarre le service par

```
start mongo
```

Le port par défaut est le 27017.
Connexion :

```
mongo --authenticationDatabase admin -u rioultf -p monPassword \
mongodb.info.unicaen.fr:27017/rioultf_bd
```

Exécution d'un script JS :

```
mongo --quiet mongodb.info.unicaen.fr:27017/rioultf_bd script.js
```

Documentation JS à l'intérieur de Mongo [2]
Le service fournit une interface web minimale sur le port 28017 (port de base +1000). Ajouter à `/etc/mongodb.conf` la ligne suivante :

```
rest = true
```

Interface de visualisation : il y a un paquet `robomongo` qui se lance dans un terminal.
Insertion depuis le terminal :

```
echo ... | mongoimport --db DATABASE_NAME --collection COLLECTION_NAME
```

Le BSON généré doit contenir un document par ligne. `jq` est fait pour cela avec son option `-c`, sinon `sed` fait aussi le boulot :

```
sed ':a;N;$!ba;s/"/\\"/g;s/\n/<br>/g' $input
```

On peut également dumper les bases par :

```
mongodump -d <db> -c <collection>
```

Cela génère un fichier BSON binaire.

2. http://ptgmedia.pearsoncmg.com/imprint_downloads/informit/learninglabs/9780133902990/ch03.html

**Liens**

- La liste des commandes : `https://docs.mongodb.com/manual/reference/mongo-shell/`
- comparaison avec SQL, une bonne option pour démarrer : `https://docs.mongodb.com/manual/reference/sql-comparison/`
- jointures en MongoDB : il faut éviter les jointures, sinon installer la 3.2 et agréger avec `$lookup`. `https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/`
- référence JavaScript : `https://docs.mongodb.com/manual/reference/method/#js-administrative-methods`

# Chapitre 5

# XML

Ce document introduit sommairement les technologies XML (eXented Markup Language), dédiées à la manipulation de documents structurés avec DTD, XMLSchema, XPath, XSL et XQuery.

Par rapport aux bases de données, ces technologies permettent de manipuler des données complexes à l'aide de procédures de transformation de haut niveau. Cependant, dans le cadre de grandes dimensions, d'opérations de jointures complexes, on préfèrera utiliser les bases de données traditionnelles.

Tout comme MongoDB gère des arbres JSON, on trouve BaseX, qui gère des arbres XML. Un langage de requêtes spécifique est utilisé : XQuery. À la différence de MongoDB, qui retourne des documents JSON dans le format où ils sont stockés dans la collection, la requête XQuery *construit* de toutes pièces le résultat en XML. Enfin, le concept de collection est absent, BaseX stocke plutôt un gigantesque arbre sur lequel on fait des requêtes avec XPath, puis on construit l'arbre XML retourné.

Les exemples sont pour la plupart tirés du site `http://www.w3schools.com/`, qui fournissent une somme complète de tutoriels et références.

## 5.1 Format XML

Un document XML représente un arbre, en imbriquant des balises qui définissent des *éléments*, pourvus d'*attributs*. Cet arbre a une racine, chaque élément a des frères, des parents et des enfants.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book category="COOKING">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
```

```
<book category="CHILDREN">
...
</book>
...<script> <![CDATA[
function matchwo(a,b){
if (a < b && a < 0) then return 1; else return 0;
}
]]>
</script>
...
</bookstore>
```

XML est simplement une norme de stockage des données, très libre dans son écriture et sa structuration de l'information. De multiples outils et normes permettent ensuite de manipuler les formats XML : DTD, XMLSchema, XPath, XSL, XQuery.

## 5.2 DTD, XMLSchema

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

La DTD (Document Type Definition) permet de préciser la structure et le type des éléments.

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

Par rapport à DTD, XMLSchema a l'avantage d'être un document XML, de mutualiser les types par le biais d'espaces de noms, et précise la structure attendue du document.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
<xs:element name="note">
<xs:complexType>
<xs:sequence>
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
<xs:element name="heading" type="xs:string"/>
<xs:element name="body" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>
```

Le traitement d'un document XML commence par son chargement. Les librairies utilisées pour cela vérifient que le document est bien formé, et s'il dispose d'une DTD ou d'un schema, vérifie que les données satisfont les contraintes.

## 5.3 XPath

XPath permet de définir les moyens d'accès aux éléments du document :

| Expression | Description |
|---|---|
| nodename | enfants du noeud |
| / | root |
| // | requête dans les fils du noeud |
| . | noeud courant |
| .. | noeud parent |
| @ | sélection d'un attribut |

Par exemple :

```
bookstore//book
//@lang
```

L'arbre retourné par un chemin XPath est une liste de noeud, dénommée *séquence*. XPath contient donc des mécanismes pour définir un chemin mais également pour manipuler une séquence : insertion, sélection, destruction, etc.

Les expressions peuvent être contraintes :

```
/bookstore/book[1]
/bookstore/book[last()]
/bookstore/book[last()-1]
/bookstore/book[position()<3]
//title[@lang]
//title[@lang='eng']
/bookstore/book[price>35.00]
/bookstore/book[price>35.00]/title
```

## 5.4 XSL

eXtented Stylesheet Language permet de transformer les données en les sélectionnant à l'aide de XPath puis en générant une transformation fondée sur des modèles.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

XSL est un langage à part entière car il peut évaluer des expressions et faire des boucles. Il y a cependant peu de mots-clé :

```
template
<xsl:for-each select="...">
<xsl:value-of select="..."/>
<xsl:sort select="..."/>
<xsl:if test="expression"> . Il n'y a pas de else, il faut utiliser choose p
<xsl:choose>
<xsl:when test="expression">
...
</xsl:when>
<xsl:otherwise>
...
</xsl:otherwise>
```

```
    </xsl:choose>
```

XSL transforme traditionnellement du XML en XML, mais peut aussi générer du texte.

## 5.5 XQuery

XQuery est le langage d'interrogation d'un ensemble de documents XML, comparable à SQL pour les bases de données. Cependant, c'est un vrai langage, avec des variables, et des fonctions. Il permet d'écrire très simplement des requêtes complexes.

```
declare function local:code(
    $headerHero as element(), $hero as element())
  as element()* {

    for $i in $headerHero/value[. != "comment(s)"]/@id
        return element {data($headerHero/value[@id=$i])}
                       {data($hero/value[@id=$i])}
};


let $root := /root
let $heroChar := $root/element[value="hero"]
let $headerHero := $root/element[id="unitBalanceID"]
(:return $headerHero:)
let $i := $headerHero
for $hero in $heroChar[value[@id="1"]](:="Hamg"]:)
let $id := $hero/value[@id="1"]
let $item := $root/file/item[@id=$id]
let $code := local:code($headerHero, $hero)
return <hero>{$id}{$item}{$code}</hero>
```

## 5.6 Utiliser XML

Il existe de nombreux outils et langages pour utiliser les technologies XML. Leur choix dépend de l'environnement, selon que l'on souhaite effectuer des batch en ligne de commande, exploiter les techniques au sein de langages évolués ou employer des environnements graphiques plus ergonomiques.

### 5.6.1 En ligne de commande

— DOMPrint pour la validation, que l'on trouve dans Xerces, la librairie d'Apache pour C++, java et Perl.
— xalan pour la transformation XSL (paquet linux)
— galax (très pénible à installer) pour XQuery

Dans les langages de haut niveau, différentes librairies sont disponibles, selon les modèles suivants :

— DOM (Document objet model) : le document XML est considéré comme un arbre, chargé intégralement, au sein duquel on peut naviguer
— SAX (Simple API for XML) : le document est lu séquentiellement. Chaque élément peut déclencher un événement.

### 5.6.2 En environnement graphique

— BaseX fournit une interface simple pour effectuer des requêtes XQuery (de donc XPath)
— eXist installe un serveur web de bases de données XML avec utilisateurs, droits, et script dynamiques en XQuery.