

Introduction au NoSQL



STÉPHANE CROZAT (ET CONTRIBUTEURS)

Table des matières

I - Introduction aux bases de données non-relationnelles	4
A. Cours.....	4
1. Perspective technologique et historique : forces et faiblesses du relationnel.....	4
2. Au delà des bases de données relationnelles : Data warehouse, XML et NoSQL.....	5
3. Bases de données NoSQL.....	7
4. Un exemple : Modélisation logique arborescente et objet en JSON.....	12
B. Exercice.....	16
1. Modélisation orientée document avec JSON.....	16
C. Devoir.....	17
1. Document sous licence Creative Commons.....	17
II - Imbrication avec Json et Mongo (base de données orientée document)	19
A. Cours.....	19
1. Exemple de base de données orientée document avec MongoDB.....	19
2. Interroger Mongo en JavaScript.....	22
B. Exercice.....	22
1. Au ciné avec Mongo.....	22
III - Référence d'objets avec Neo4J (BD orientée graphe)	27
A. Cours.....	27
1. Exemple de base de données orientée graphe avec Neo4J.....	27
B. Exercice.....	29
1. De Neo4J à Game of Thrones.....	29
IV - Modélisation logique arborescente en XML	35
A. Cours.....	35
1. Introduction à XML.....	35
2. Syntaxe de base XML.....	40
3. Introduction aux schémas XML.....	43
4. Manipulation XML avec XPath.....	49
B. Exercice.....	51
1. Mon nom est personne.....	51
2. Glossaire I.....	52
C. Devoir.....	53
1. On l'appelle Trinita.....	53

b) 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 supprimer 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 le problème posé par le décisionnel.

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

a) Problème de l'agrégat et développement des data warehouses

Fondamental

Le modèle relationnel est peu performant pour les agrégats.

Problème posé par le décisionnel et résolu par les data warehouses

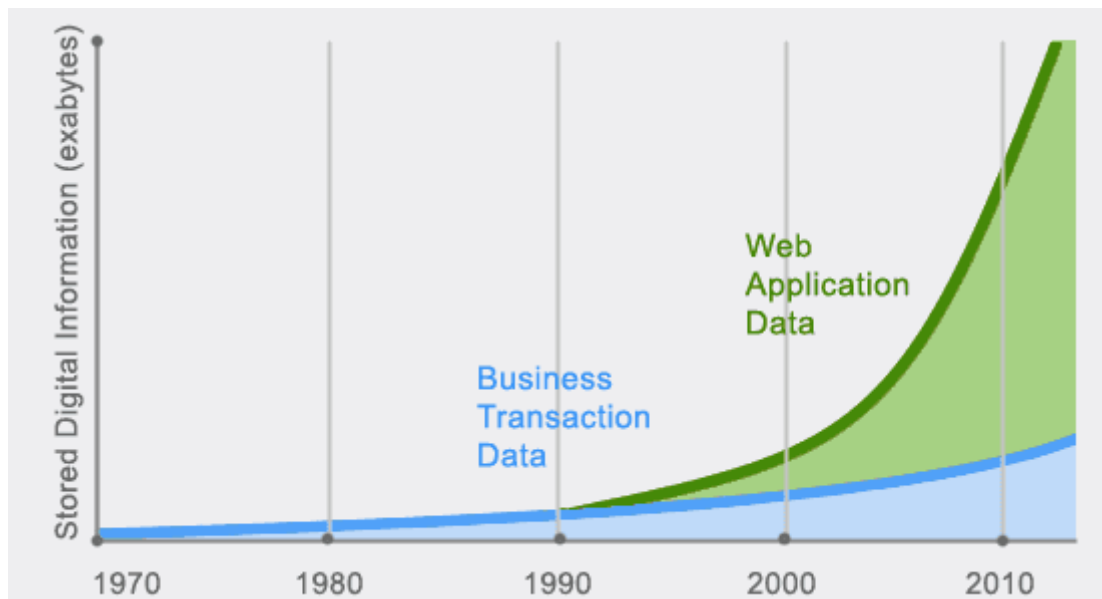
- décision vs gestion
- agrégat vs accès ciblé
- historisation vs transaction

b) Problème de l'impedance mismatch et développement de l'objet et de XML

Fondamental

OID et nested model

c) Problème de la distribution et développement du NoSQL

Big data

Évolution des volumes de données d'entreprise versus web

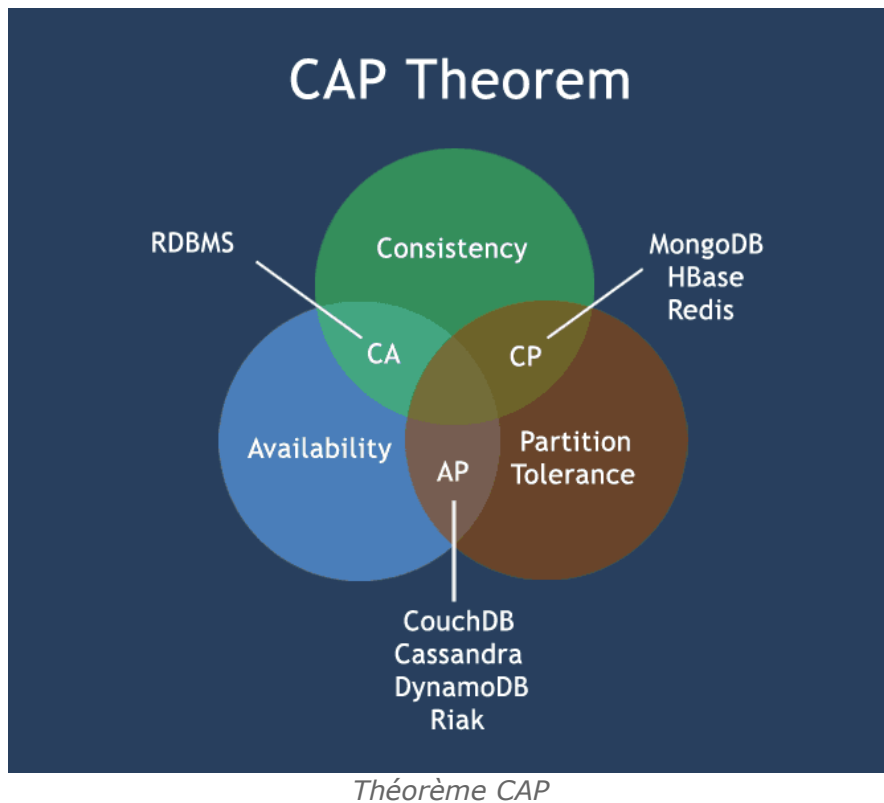
Fondamental : Distribution

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*

Fondamental : Le "commerce" de la 3NF

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



3. Bases de données NoSQL

a) 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/>¹



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

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

Complément

<http://blog.xebia.fr/2010/04/21/nosql-europe-tour-dhorizon-des-bases-de-donnees-nosql/>²

b) Fondamentaux des modèles NoSQL : Clé-valeur, distribution, imbrication, schema-less

Fondamental : Sharding et clé-valeur

- **Simplification du modèle en clé-valeur**
- **Distribution sur les nœuds d'un cluster**

Identification

Object Identifiers (OID)

Universally Unique Identifier (UUID)

Uniform Resource Name (URN)

Imbrication

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

Hachage et distribution

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

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.

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

c) Illustration des modèles NoSQL

*Exemple : Représentation de ventes en relationnel***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

*Exemple : Représentation de ventes en colonne***Family Sales**

#ticket							
1	<table><tr><th>date</th></tr><tr><td>01/01/16</td></tr></table>	date	01/01/16	<table><tr><th>books</th></tr><tr><td>2212121504</td></tr><tr><td>2212141556</td></tr></table>	books	2212121504	2212141556
date							
01/01/16							
books							
2212121504							
2212141556							
2	<table><tr><th>date</th></tr><tr><td>01/01/16</td></tr></table>	date	01/01/16	<table><tr><th>books</th></tr><tr><td>2212141556</td></tr></table>	books	2212141556	
date							
01/01/16							
books							
2212141556							

Family Book

#isbn									
2212121504	<table><tr><th>title</th></tr><tr><td>Scenari</td></tr></table>	title	Scenari						
title									
Scenari									
2212141556	<table><tr><th>title</th></tr><tr><td>NoSQL</td></tr></table>	title	NoSQL	<table><tr><th>a-surname</th></tr><tr><td>Bruchez</td></tr></table>	a-surname	Bruchez	<table><tr><th>a-firstname</th></tr><tr><td>Rudi</td></tr></table>	a-firstname	Rudi
title									
NoSQL									
a-surname									
Bruchez									
a-firstname									
Rudi									

*Exemple : Représentation de ventes en document***Collection Sales**

#oid	
4d040766076 6b236450b45 a3	"ticket" : 1 "date" : "01/01/16" "books" : ["_id" : 2212121504 "_id" : 2212141556]
4d040766076 6b236450b45 a4	"ticket" : 2 "date" : "01/01/16" "books" : ["_id" : 2212141556]

Collection Book

#oid	
4d040766076 6b236450b45 a5	"isbn" : 2212121504 "title" : "Scenari"
4d040766076 6b236450b45 a6	"isbn" : 2212141556 "title" : "NoSQL" "author" : { "surname" : Bruchez "firstname" : Rudi }

Exemple : Représentation de ventes en document (avec imbrication redondante)

Collection Sales

#oid	
4d040766076 6b236450b45 a3	<pre> "ticket" : 1 "date" : "01/01/16" "books" : [{ "isbn" : 2212121504 "title" : "Scenari" } { "isbn" : 2212141556 "title" : "NoSQL" "author" : { "surname" : Bruchez "firstname" : Rudi } }]</pre>
4d040766076 6b236450b45 a4	<pre> "ticket" : 2 "date" : "01/01/16" "books" : [{ "isbn" : 2212141556 "title" : "NoSQL" "author" : { "surname" : Bruchez "firstname" : Rudi } }]</pre>

Exemple : Représentation de ventes en graphe

Classe Sales

#oid		
4d040766076 6b236450b45 a3	property	ticket : 1
	property	date : 01/01/16
	relation	book : 4d0407660766b236450b45a5
	relation	book : 4d0407660766b236450b45a6
4d040766076 6b236450b45 a4	property	ticket : 2
	property	date : 01/01/16
	relation	book : 4d0407660766b236450b45a6

Classe Book

#oid		
4d040766076 6b236450b45 a5	property	title : Scenari
4d040766076 6b236450b45 a6	property	title : NoSQL
	relation	author : 4d0407660766b236450b45a8

Classe Author

#oid		
4d040766076 6b236450b45 a8	property	surname : Bruchez
	property	firstnam : Rudi

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

a) JavaScript Object Notation

Définition : Introduction

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).

Exemple : Un fichier JSON simple

```

1 {
2   "nom" : "Norris",
3   "prenom" : "Chuck",
4   "age" : 73,
5   "etat" : "Oklahoma"
6 }
```

Attention : JSON est Indépendant de tout langage

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*.

Complément : Extension

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

b) La syntaxe JSON en bref

Syntaxe : Règles syntaxiques

- 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.

Fondamental : Éléments du format JSON

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.

Définition : Valeurs possibles

- 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

```

1 {
2   "nom cours" : "NF29",
3   "theme" : "ingenierie documentaire",
4   "etudiants" : [
5     {
6       "nom" : "Norris",
7       "prenom" : "Chuck",
8       "age" : 73,
9       "pays" : "USA"
10    },
11    {
12      "nom" : "Doe",
13      "prenom" : "Jane",
14      "age" : 45,
15      "pays" : "Angleterre"
16    },
17    {
18      "nom" : "Ourson",
19      "prenom" : "Winnie",
20      "age" : 10,
21      "pays" : "France"
22    }
23  ]
24 }
```

c) 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.

Exemple : APIs retournant des données au format JSON

Twitter : <https://dev.twitter.com/rest/public>³ : récupération de données du réseau social.

Netatmo : <https://dev.netatmo.com/doc/publicapi>⁴ : 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.

Exemple : Fonctions JSON de Postgres et MySQL

Fonctions PostgreSQL : <http://www.postgresql.org/docs/9.3/static/functions-json.html>⁵

Fonctions MySQL : <https://dev.mysql.com/doc/refman/5.7/en/json.html>⁶

d) 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) ?

3 - <https://dev.twitter.com/rest/public>

4 - <https://dev.netatmo.com/doc/publicapi>

5 - <http://www.postgresql.org/docs/9.3/static/functions-json.html>

6 - <https://dev.mysql.com/doc/refman/5.7/en/json.html>

- ☐ 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

e) Un programme 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>Démon 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>

```

```

1 function query() {
2   // File
3   var vFile = document.getElementById("myfile").files[0];
4   // Reader
5   var vReader = new FileReader();
6   vReader.readAsText(vFile);
7   vReader.onload = function(pEvent) {
8     // String Input
9     var vContent = pEvent.target.result;
10    // JSON to object
11    var vJson = JSON.parse(vContent);
12    // Query
13    var vResult = vJson.prenom + " " + vJson.nom + " (" + vJson.age + ")";
14    // Output
15    document.getElementById("mydiv").appendChild(document.createTextNode(vResult));
16  };
17 }

```

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

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 parcourt le tableau JSON à l'aide d'une boucle.

```

1  for (var i=0;i<vJson.adresse.length;i++) {
2      ...
3  }
```

B. Exercice

1. 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.

```

1  {
2    "code":"api04",
3    "titre":"DW et NOSQL"
4    ...
```


5 }

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.

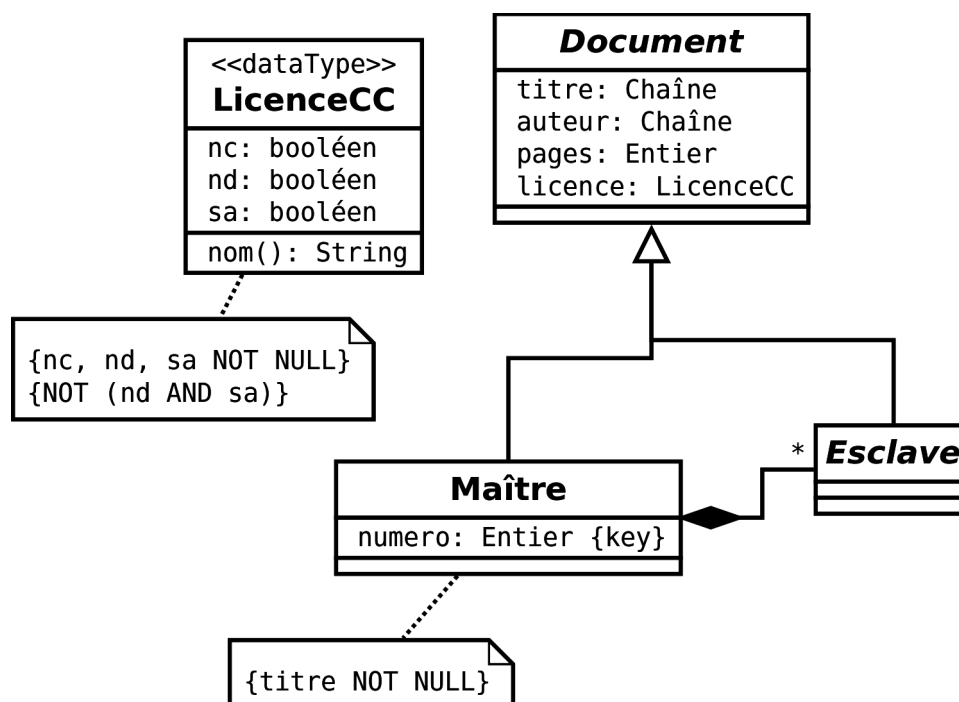
C. Devoir

1. Document sous licence Creative Commons

On souhaite créer un langage permettant de d'assembler des documents situés dans une base de données. 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 maître, il dispose alors toujours d'un numéro d'enregistrement unique dans la base, et son titre sera toujours renseigné ;
- soit un document esclave, il est alors membre d'un document maître.



Question 1

Proposez une représentation JSON des données (le document numéro 1 et ses deux documents esclaves) en vue d'expérimenter une solution MongoDB.

Indice :

On notera que le numéro d'enregistrement doit être intégré au JSON, c'est une clé naturelle, puisqu'il fait partie du cahier des charges (il n'est pas ajouté au moment de l'implémentation technique, mais dès la modélisation UML).

Question 2

Donner un avantage et un inconvénient de cette approche non-relationnelle par rapport à une approche relationnelle. Vous n'avez le droit qu'à une seule phrase précise et synthétique pour l'avantage et une autre pour l'inconvénient, chacune s'appuyant sur le cas précis de l'exercice.

1. Exemple de base de données orientée document avec MongoDB

Le serveur MongoDB est organisé en plusieurs *databases* :

- | SQL | MongoDB |
|------------------------------|--|
| base de données et/ou schéma | base de données |
| table | collection |
| enregistrement | document |
| attribut (atomique) | propriété (chaîne, entier, tableau, structure) |

Year	Percentage
1990	10%
1992	15%
1994	20%
1996	25%
1998	30%
2000	35%
2002	40%
2004	45%
2006	50%
2008	45%
2010	55%

Schema-less

C'est une base *schema-less*, aussi une collection peut contenir des documents de structures différentes et il n'est pas possible de définir la structure a priori d'une collection. La structure d'une collection n'est donc définie que par les document qui la compose, et elle peut évoluer dynamiquement au fur et à mesure des insertions et suppressions.

Identification clé / valeur

Chaque document est identifié par un identifiant nommé `_id` unique pour une collection, fonctionnant comme une **clé primaire artificielle**.

Architecture

MongoDB fonctionne a minima sous la forme d'un serveur auquel il est possible de se connecter avec un client textuel (*mongo shell*).

MongoDB peut être distribuée sur plusieurs serveurs (partitionnement horizontal ou *sharding*) et accédée à travers de multiples couches applicatives (langages, API...)

Complément

<https://docs.mongodb.org/manual/>⁷

b) Installation et démarrage de MongoDB

MongoDB est disponible sur Windows, OSX et Linux.

<https://docs.mongodb.com/manual/installation/>⁸

Client textuel Mongo (mongo shell)

Pour se connecter à un serveur MongoDB :

- présent sur la même machine : exécuter simplement `mongo` dans un terminal ;
- distant sur le port standard de MongoDB : exécuter `mongo --host nom-du-serveur`.

Test

Une fois connecté exécuter le code suivant pour vérifier le bon fonctionnement de la base :

```
1 db.test.insert({ "a":0 })
2 db.test.find()
```

Le résultat attendu est le suivant, la clé générée étant bien entendu différente :

```
{ "_id":ObjectId("574ebe32b3286a9a8fadb55"), "a":0 }
```

Complément

<https://docs.mongodb.com/manual/mongo/>⁹

c) Créer des bases de données et des collections

Fondamental : Créer une base et une collection

MongoDB est une base *schema-less*, la création des bases et des collections est dynamique lors d'une première **insertion** de données.

Méthode

Pour créer une base de données il faut exécuter une instruction `use` sur une nouvelle base de données, puis donner un ordre d'insertion d'un premier document JSON avec `insert`.

7 - <https://docs.mongodb.org/manual>

8 - <https://docs.mongodb.com/manual/installation/>

9 - <https://docs.mongodb.com/manual/mongo>

Exemple

```
use dbl
db.coll.insert( { "x":1 } )
```

Syntaxe : Catalogue de données

- On peut voir la liste des bases de données avec :
`show dbs`
- On peut voir la liste des collections de la base de données en cours avec :
`show collections`
- On peut voir le contenu d'une collection avec :
`db.coll.find()`

Complément

<https://docs.mongodb.com/manual/mongo>

d) Insertion des documents

L'insertion de données dans une base MongoDB se fait avec l'instruction `db.collection.insert(Document JSON)`.

Si la collection n'existe pas, elle est créée dynamiquement.

L'insertion associe un identifiant (`_id`) à chaque document de la collection.

Exemple

```
1 db.Cinema.insert(
2 {
3   "nom":"Honkytonk Man",
4   "realisateur":{
5     "nom":"Eastwood",
6     "prenom":"Clint"
7   },
8   "annee":1982,
9   "acteurs":[
10    {
11      "nom":"Eastwood",
12      "prenom":"Kyle"
13    },
14    {
15      "nom":"Eastwood",
16      "prenom":"Clint"
17    }
18  ]
19 }
20 )
```

Complément

<https://docs.mongodb.org/manual/core/crud>¹⁰

<https://docs.mongodb.com/manual/tutorial/insert-documents>¹¹

e) Trouver des documents

La recherche de données dans une base MongoDB se fait avec l'instruction `db.collection.find(Document JSON, document JSON)`, avec :

- le premier document JSON définit une restriction ;
- le second document JSON définit une projection (ce second argument est optionnel).

10 - <https://docs.mongodb.org/manual/core/crud/>

11 - <https://docs.mongodb.com/manual/tutorial/insert-documents/>

Exemple : Restriction

```
1 db.Cinema.find({"nom":"Honkytonk Man"})
```

retourne les document JSON tels qu'ils ont à la racine un attribut "nom" avec la valeur "Honkytonk Man".

Exemple : Restriction et projection

```
1 db.Cinema.find({"nom":"Honkytonk Man"}, {"nom":1, "realisateur":1} )
```

retourne les document JSON tels qu'ils ont à la racine un attribut "nom" avec la valeur "Honkytonk Man", et seul les attributs situés à la racine "nom" et "realisateur" sont projetés (en plus de l'attribut "_id" qui est projeté par défaut).

Complément

<https://docs.mongodb.org/manual/tutorial/query-documents/>¹²

2. Interroger Mongo en JavaScript

a) Interroger Mongo en JavaScript

Le console mongo permet d'exécuter des programme JavaScript avec instruction load.

```
1 //test.js
2 print("Hello world");
```

```
1 > load("test.js")
```

Parcours d'un résultat de requête Mongo

```
1 //query.js
2 conn = new Mongo();
3 db = conn.getDB("db1");
4
5 recordset = db.User.find({"liked":{"$elemMatch":{"star":3}}}, {"_id":0,
6   "liked.film":1})
7
8 while ( recordset.hasNext() ) {
9   printjson( recordset.next() );
10 }
```

Complément

<https://docs.mongodb.com/manual/tutorial/write-scripts-for-the-mongo-shell/>¹³

B. Exercice

1. Au ciné avec Mongo

[1h30]

Soit les données suivantes représentant des films de cinéma.

12 - <https://docs.mongodb.org/manual/tutorial/query-documents/>

13 - <https://docs.mongodb.com/manual/tutorial/write-scripts-for-the-mongo-shell/>

```

1 db.Cinema.drop()
2
3 db.Cinema.insert(
4 {
5   nom:"Goodfellas",
6   annee:1990,
7   realisateur:{nom:"Scorsese", prenom:"Martin"},
8   acteurs:
9   [
10    {nom:"De Niro", prenom:"Robert"},
11    {nom:"Liotta", prenom:"Ray"},
12    {nom:"Pesci", prenom:"Joe"}
13   ]
14 })
15
16 db.Cinema.insert(
17 {
18   nom:"The Godfather",
19   annee:1972,
20   realisateur:{nom:"Coppola", prenom:"Francis Ford"},
21   acteurs:
22   [
23    {nom:"Pacino", prenom:"Al"},
24    {nom:"Brando", prenom:"Marlon"},
25    {nom:"Duvall", prenom:"Robert"}
26   ]
27 })
28
29 db.Cinema.insert(
30 {
31   nom:"Million Dollar Baby",
32   realisateur:{nom:"Eastwood", prenom:"Clint"},
33   acteurs:
34   [
35    {nom:"Swank", prenom:"Hilary"},
36    {nom:"Eastwood", prenom:"Clint"}
37   ]
38 })
39
40 db.Cinema.insert(
41 {
42   nom:"Gran Torino",
43   annee:2008,
44   realisateur:{nom:"Eastwood", prenom:"Clint"},
45   acteurs:
46   [
47    {nom:"Vang", prenom:"Bee"},
48    {nom:"Eastwood", prenom:"Clint"}
49   ]
50 })
51
52 db.Cinema.insert(
53 {
54   nom:"Unforgiven",
55   realisateur:{nom:"Eastwood", prenom:"Clint"},
56   acteurs:
57   [
58    {nom:"Hackman", prenom:"Gene"},
59    {nom:"Eastwood", prenom:"Clint"}
60   ]
61 })
62
63 db.Cinema.insert(
64 {
65   nom:"Mystic River",
66   realisateur:{nom:"Eastwood", prenom:"Clint"},
67   acteurs:
68   [

```

```

69     {nom:"Penn", prenom:"Sean"},
70     {nom:"Bacon", prenom:"Kevin"}
71   ]
72 })
73
74 db.Cinema.insert(
75   {
76     nom:"Honkytonk Man",
77     realiseur:{nom:"Eastwood", prenom:"Clint"},
78     annee:1982,
79     acteurs:
80     [
81       {nom:"Eastwood", prenom:"Kyle"},
82       {nom:"Bloom", prenom:"Verna"}
83     ]
84   })
85
86 db.Cinema.find()

```

L'objectif est d'initialiser une base MongoDB avec ce script, puis d'écrire les requêtes MongoDB permettant de répondre aux questions suivantes.

Question 1

Créer une nouvelle base MongoDB et exécuter le script. Nommez votre base par votre nom de famille ou votre login sur la machine par exemple.

Indices :

Pour créer une base de données, utiliser l'instruction `use myNewDatabase`, puis exécuter au moins une instruction d'insertion.

Créer des bases de données et des collections

Question 2

Quels sont les films sortis en 1990 ?

Indices :

Trouver des documents

`db.Cinema.find(document JSON)`

La syntaxe JSON

`db.col.find({"attribute":"value"})`

Question 3

Quels sont les films sortis avant 2000 ?

Indice :

On utilisera l'objet `{ $lt: value }` à la place de la valeur de l'attribut à tester (`$lt` pour lesser than).

Question 4

Quels sont les films réalisés par Clint Eastwood ?

Indice :

On utilisera un objet comme valeur.

Question 5

Quels sont les films réalisés par quelqu'un prénommé Clint ?

Indice :

Utiliser le navigateur de propriété des objets point : `object.attribute`.

Question 6

Quels sont les films réalisés par quelqu'un prénommé Clint avant 2000 ?

Indice :

Utiliser une liste de conditions attribut:valeur pour spécifier un AND (et logique) :
`db.col.find({"attribute1":"value1", "attribute2":"value2"})`

Question 7

Quels sont les films dans lesquels joue Clint Eastwood ?

Question 8

Quels sont les films dans lesquels joue un Eastwood ?

Indice :

Utiliser le parser de tableau `$elemMatch:{"key":"value"}` à la place de la valeur.

Question 9

Quels sont les noms des films dans lesquels joue un Eastwood ?

Indices :

*Pour gérer la **projection**, utiliser un second argument de la clause `find()` :*
`db.Cinema.find({document JSON de sélection }, {document JSON de projection})`
avec document JSON de projection de la forme : `{"attribut1":1, "attribut2":1...}`
Les identifiants sont toujours affichés par défaut, si on veut les supprimer, on peut ajouter la clause `_id:0` dans le document de projection.

Question 10

Compléter le programme JavaScript suivant afin d'afficher les titre selon le format suivant :

```
1 - Million Dollar Baby
2 - Gran Torino
3 - Unforgiven
4 - Honkytonk Man
```

Indice :

```
1 conn = new Mongo();
2 db = conn.getDB("...");
3
4 recordset = ...
5
6 while ( recordset.hasNext() ) {
7     film = recordset.next() ;
8     print("- ", ...);
9 }
```

On veut à présent ajouter une nouvelle collection permettant de gérer des utilisateurs et leurs préférences. Pour chaque utilisateur on gèrera un pseudonyme, et une liste de films préférés avec une note allant de une à trois étoiles.

Question 11

Ajouter trois utilisateurs préférant chacun un ou deux films.

On utilisera les identifiants des films pour les référencer.

Indice :

Insertion des documents

Question 12

Critiquer cette insertion mobilisant les identifiants des films ? Pourquoi n'est ce pas reproductible ?

Question 13

Quels sont les utilisateurs qui aiment au moins un film avec 3 étoiles ?

Question 14

Quels sont les identifiants des films qui sont aimés au moins une fois avec 3 étoiles ?

Question 15

Pourquoi trouver les titres de ces films n'est pas simple avec Mongo ?

Question 16

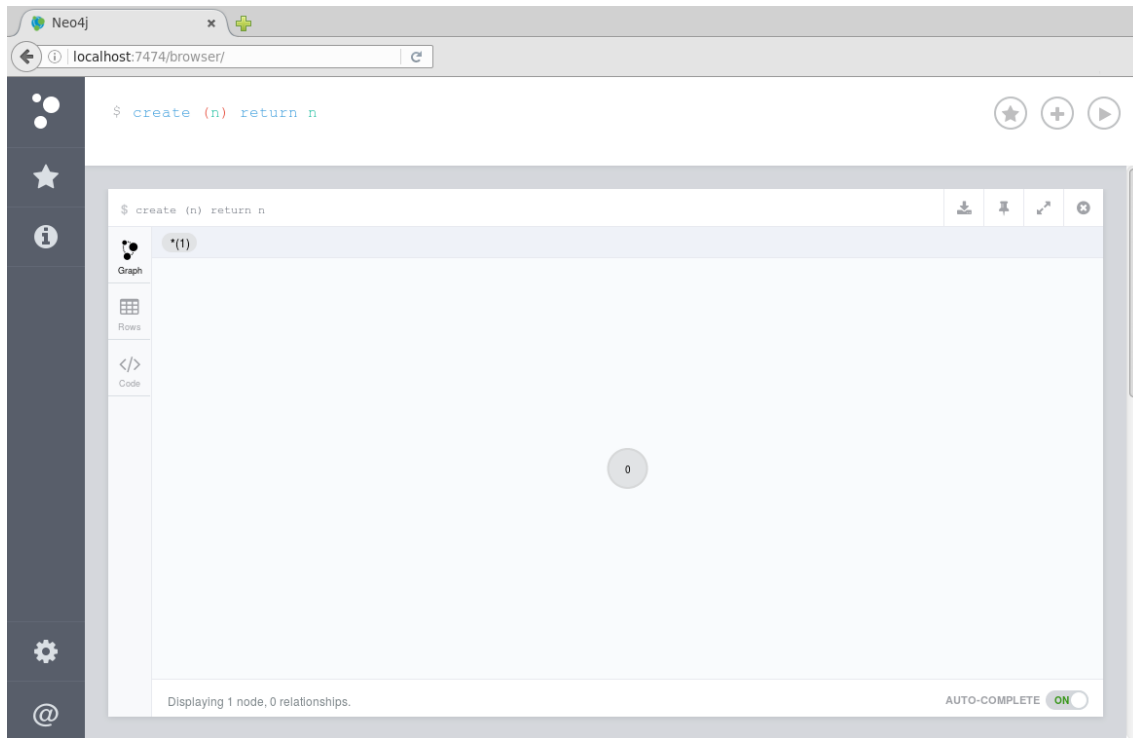
Écrire le programme JavaScript permettant de récupérer la liste dédoublonnée des identifiants de films qui sont aimés au moins une fois avec 3 étoiles, puis d'afficher les titres des films correspondants.

Indice :

```
1  conn = new Mongo();
2  db = conn.getDB("...");
3
4  films = ...
5
6  for (var i=0; i<films.length; i++) {
7    var id = films[i];
8    if (film = ...)
9      print(...);
10 }
11
```


- Pour réinitialiser les bases de données : supprimer le dossier \$NEO4J_HOME/data/graph.db/

c) Test de Neo4J



Interface de Neo4j (création d'un nœud)

- Les requêtes s'écrivent dans la partie haute.
- Les résultats se visualisent dans la partie basse.

Exemple : Créer un nœud

```
create (n) return n
```

Exemple : Voir tous les nœuds

```
match (n) return n
```

d) Créer des nœuds et des relations

Exemple : Créer un nœud avec une propriété

```
create (n {name:'Sun'}) return n
```

Exemple : Créer un nœud avec un label

```
create (n:Astre {name:'Sun'}) return n
```

Syntaxe : Créer des relations

le formalisme pour les relations est :

- ()<-[]-()
- ()-[]->()
- ()<-[]->()
- ()-[]-()

où :

- on mettra nos nœuds à l'intérieur des parenthèses ()
- on mettra nos relations à l'intérieur des crochets []
- enfin on désignera le sens de la relation avec la flèche

Exemple : Créer deux nœuds et une relation entre ces deux noeuds

```
create (e:Astre {name:'Earth'}) <-[:Satellite {distance:384000}]- (m:Astre {name:'Moon'}) return e, m
```

Exemple : Créer deux nœuds et une relation entre ces deux noeuds en plusieurs instructions

```
create (e:Astre {name:'Earth'})
create (m:Astre {name:'Moon'})
create (e)<-[:Satellite {distance:384000}]- (m)
return e, m
```

Remarque

- CTRL+ENTER
Lance la commande présente dans la ligne de commande.
- SHIFT+ENTER
Permet de faire un saut de ligne dans la ligne de commande (sans l'exécuter)

e) Trouver des nœuds et des relations

Exemple : Trouver un noeud avec une propriété

```
match (n {name:'Sun'}) return n
```

Exemple : Trouver un nœud avec un label

```
match (n:Astre) return n
```

Exemple : Trouver des nœuds en fonction d'une relation

```
match (n1) <-[:Satellite]- (n2) return r,n1,n2
```

B. Exercice

1. De Neo4J à Game of Thrones

[1h]

Un beau matin, vous vous réveillez en Westeros, le monde de Game Of Thrones, sans aucune connaissance historique. Afin d'y survivre, il va falloir intégrer les informations du royaume. Nous allons pour cela utiliser une base de données orientée graphe Neo4J.



Question 1

Avant toute chose vous allez vous allez devoir vous ajouter au monde que vous venez de rejoindre.

Créer un nouveau personnage avec votre nom et prénom.

Indices :

Créer des nœuds et des relations

Le label est :personnage.

Les propriétés sont :

{name : '...', nickname : '...' }

Question 2

Seconde chose, il faut apprendre à créer des relations, car le monde de Game Of Thrones est très complexe.

Créez deux personnages et exprimez une relation qui les relie.

Par exemple, un des combats les plus célèbres oppose Oberyn Martell, dit The Viper et Gregor Clegane, dit The Mountain. Gregor Clegane tue Oberyn Martell en duel.

Indices :

Créer des nœuds et des relations

Créer d'abord les deux nœuds avec l'instruction CREATE.

```
1 CREATE(gregor:personnage { name : '...', nickname : '...' })
2 CREATE(oberyn:...)
3 ...
```

Toujours avec CREATE, créer une relation de type « :TUE » entre les deux nœuds avec comme attribut : « type : 'duel' »

```
1 CREATE(gregor:personnage { name : 'Gregor Clegane', nickname : 'The Mountain' })
```

```

2 CREATE(oberyn:personnage { name : 'Oberyn Martell', nickname : 'The
  Viper' })
3 CREATE(...)-[...]-(...)
4 ...

```

```

1 CREATE(gregor:personnage { name : 'Gregor Clegane', nickname : 'The
  Mountain' })
2 CREATE(oberyn:personnage { name : 'Oberyn Martell', nickname : 'The
  Viper' })
3 CREATE(gregor)-[...]-(oberyn)
4 RETURN gregor, oberyn

```

```

1 CREATE(gregor:personnage { name : 'Gregor Clegane', nickname : 'The
  Mountain' })
2 CREATE(oberyn:personnage { name : 'Oberyn Martell', nickname : 'The
  Viper' })
3 CREATE(gregor)-[:TUE { type : ... }]->(oberyn)
4 RETURN gregor, oberyn

```

Question 3

Affichez l'ensemble des nœuds présents dans votre base de données.

Indice :

Test de Neo4J

Nous pouvons à présent alimenter notre base d'un ensemble de connaissance dont nous disposons. Pour cela copier et exécuter le code Cypher ci-après.

```

1 //GameOfThrones:
2
3 //sans clan :
4 CREATE (sansclan:clan { clanname : 'Clan des sans lien de sang'})
5
6 CREATE (hodor:personnage { name : 'Hodor', nickname : 'I am the only good actor
  here !' })
7 CREATE (khal:personnage { name : 'Khal Drogo', nickname : 'Horse Man' })
8 CREATE (petyr:personnage { name : 'Petyr Baelish', nickname : 'LittleFinger' })
9 CREATE (shae:personnage { name : 'Shae for the Imp', nickname : 'Dead' })
10
11
12 //lanister :
13 CREATE (lanister:clan { clanname : 'The Lannisters'})
14
15 CREATE (tyrion:personnage { name : 'Tyrion Lannister', nickname : 'The Imp' })
16 CREATE (tywin:personnage { name : 'Tywin Lannister', nickname : 'Father of the
  Lannisters' })
17 CREATE (jaime:personnage { name : 'Jaime Lannister', nickname : 'King Slayer' })
18 CREATE (cersei:personnage { name : 'Cersei Lannister', nickname : 'Brother Lover'
  })
19 CREATE (joffrey:personnage { name : 'Joffrey Lannister', nickname : 'Mad King
  2' })
20
21
22 //baratheons :
23 CREATE (baratheon:clan { clanname : 'The Baratheons'})
24
25 CREATE (robert:personnage { name : 'Robert Baratheon', nickname : 'King
  Robert' })
26 CREATE (renly:personnage { name : 'Renly Baratheon', nickname : 'King Gay' })
27 CREATE (stannis:personnage { name : 'Stannis Baratheon', nickname : 'Stéphane' })
28 CREATE (gendry:personnage { name : 'Gendry ???Baratheon', nickname : 'What
  happened to my story ?' })
29
30

```

```

31 //tyrells :
32 CREATE (tyrell:clan { clanname : 'The Tyrells'})
33
34 CREATE (margeary:personnage { name : 'Margeary Tyrell', nickname : 'Tyrell asset'
35 })
36 CREATE (loras:personnage { name : 'Loras Tyrell', nickname : 'King Gay Knight' })
37
38 //targaryens :
39 CREATE (targaryen:clan { clanname : 'The Targaryen'})
40
41 CREATE (daenerys:personnage { name : 'Daenerys Targaryen', nickname : 'Mother of
42 Dragons' })
43 CREATE (viserys:personnage { name : 'Viserys Targaryen', nickname : 'Gold
44 Head' })
45
46 //Stark :
47 CREATE (stark:clan { clanname : 'The Starks'})
48
49 CREATE (arya:personnage { name : 'Arya Stark', nickname : 'I am 20' })
50 CREATE (sansa:personnage { name : 'Sansa Stark', nickname : 'I am sorry I can t
51 smile.' })
52 CREATE (rosse:personnage { name : 'Roose Bolton', nickname : 'Come by the house I
53 ll kill you' })
54 CREATE (edward:personnage { name : 'Eddard Stark', nickname : 'Whhhhhy ???' })
55 CREATE (robb:personnage { name : 'Robb Stark', nickname : 'King of the North' })
56 CREATE (john:personnage { name : 'John Snow', nickname : 'The Bastard' })
57 CREATE (bran:personnage { name : 'Bran Stark', nickname : 'Stop boring me' })
58
59 //tullys :
60 CREATE (tully:clan { clanname : 'The Tullys'})
61
62 CREATE (catelyn:personnage { name : 'Catelyn Stark', nickname : 'Mother of
63 Stark ?' })
64 CREATE (lysa:personnage { name : 'Lyse Tully', nickname : 'Crazy' })
65
66 //greyjoys :
67 CREATE (greyjoy:clan { clanname : 'The Greyjoys'})
68
69 CREATE (theon:personnage { name : 'Theon Greyjoy', nickname : 'The enuque' })
70 CREATE (balon:personnage { name : 'Balon Greyjoy', nickname : 'Nicest father
71 ever' })
72
73 //actions :
74
75 CREATE (tyrion)-[:TUE{type : 'homicide'}]->(shae)
76 CREATE (tyrion)-[:TUE{type : 'homicide'}]->(tywin)
77 CREATE (petyr)-[:TUE{type : 'homicide'}]->(joffrey)
78 CREATE (stannis)-[:TUE{type : 'homicide'}]->(renly)
79 CREATE (khal)-[:TUE{type : 'homicide'}]->(khal)
80 CREATE (khal)-[:TUE{type : 'homicide'}]->(viserys)
81 CREATE (joffrey)-[:TUE{type : 'homicide'}]->(edward)
82 CREATE (rosse)-[:TUE{type : 'homicide'}]->(robb)
83 CREATE (rosse)-[:TUE{type : 'homicide'}]->(catelyn)
84 CREATE (petyr)-[:TUE{type : 'homicide'}]->(lysa)
85
86 CREATE (jaime)-[:AIME{type : 'amour'}]->(cersei)
87 CREATE (cersei)-[:AIME{type : 'amour'}]->(jaime)
88 CREATE (tyrion)-[:AIME{type : 'amour'}]->(shae)
89 CREATE (shae)-[:AIME{type : 'amour'}]->(tywin)
90 CREATE (robert)-[:AIME{type : 'amour'}]->(cersei)
91 CREATE (loras)-[:AIME{type : 'amour'}]->(renly)
92 CREATE (renly)-[:AIME{type : 'amour'}]->(loras)
93 CREATE (margeary)-[:AIME{type : 'amour'}]->(joffrey)
94 CREATE (joffrey)-[:AIME{type : 'amour'}]->(margeary)

```



```

93 CREATE (khal)-[:AIME{type : 'amour'}]->(daenerys)
94 CREATE (daenerys)-[:AIME{type : 'amour'}]->(khal)
95 CREATE (petyr)-[:AIME{type : 'amour'}]->(catelyn)
96 CREATE (edward)-[:AIME{type : 'amour'}]->(catelyn)
97 CREATE (catelyn)-[:AIME{type : 'amour'}]->(edward)
98
99
100 //liens de clan :
101
102 CREATE (theon)-[:LIER{type : 'liendecan'}]->(greyjoy)
103 CREATE (balon)-[:LIER{type : 'liendecan'}]->(greyjoy)
104
105 CREATE (khal)-[:LIER{type : 'liendecan'}]->(sansclan)
106 CREATE (john)-[:LIER{type : 'liendecan'}]->(sansclan)
107 CREATE (petyr)-[:LIER{type : 'liendecan'}]->(sansclan)
108 CREATE (gendry)-[:LIER{type : 'liendecan'}]->(sansclan)
109 CREATE (hodor)-[:LIER{type : 'liendecan'}]->(sansclan)
110
111 CREATE (gendry)-[:LIER{type : 'liendecan'}]->(baratheon)
112 CREATE (joffrey)-[:LIER{type : 'liendecan'}]->(baratheon)
113 CREATE (robert)-[:LIER{type : 'liendecan'}]->(baratheon)
114 CREATE (renly)-[:LIER{type : 'liendecan'}]->(baratheon)
115 CREATE (stannis)-[:LIER{type : 'liendecan'}]->(baratheon)
116
117 CREATE (margary)-[:LIER{type : 'liendecan'}]->(tyrell)
118 CREATE (loras)-[:LIER{type : 'liendecan'}]->(tyrell)
119
120 CREATE (daenerys)-[:LIER{type : 'liendecan'}]->(targaryen)
121 CREATE (viserys)-[:LIER{type : 'liendecan'}]->(targaryen)
122
123 CREATE (lysa)-[:LIER{type : 'liendecan'}]->(tully)
124 CREATE (catelyn)-[:LIER{type : 'liendecan'}]->(tully)
125
126 CREATE (arya)-[:LIER{type : 'liendecan'}]->(stark)
127 CREATE (hodor)-[:LIER{type : 'liendecan'}]->(stark)
128 CREATE (rosse)-[:LIER{type : 'liendecan'}]->(stark)
129 CREATE (sansa)-[:LIER{type : 'liendecan'}]->(stark)
130 CREATE (petyr)-[:LIER{type : 'liendecan'}]->(stark)
131 CREATE (edward)-[:LIER{type : 'liendecan'}]->(stark)
132 CREATE (robb)-[:LIER{type : 'liendecan'}]->(stark)
133 CREATE (john)-[:LIER{type : 'liendecan'}]->(stark)
134 CREATE (bran)-[:LIER{type : 'liendecan'}]->(stark)
135 CREATE (catelyn)-[:LIER{type : 'liendecan'}]->(stark)
136 CREATE (theon)-[:LIER{type : 'liendecan'}]->(stark)
137
138 CREATE (shae)-[:LIER{type : 'liendecan'}]->(lanister)
139 CREATE (rosse)-[:LIER{type : 'liendecan'}]->(lanister)
140 CREATE (tyrion)-[:LIER{type : 'liendecan'}]->(lanister)
141 CREATE (tywin)-[:LIER{type : 'liendecan'}]->(lanister)
142 CREATE (jaime)-[:LIER{type : 'liendecan'}]->(lanister)
143 CREATE (cersei)-[:LIER{type : 'liendecan'}]->(lanister)
144 CREATE (joffrey)-[:LIER{type : 'liendecan'}]->(lanister)

```

Question 4

Affichez l'ensemble des nœuds présents dans votre base de données.
Manipulez le graphe quelques minutes afin de vous faire une idée des données.

Question 5

Affichez à présent la liste des clans.

Indice :

On utilise le label :clan après l'alias de nœud n.

Trouver des nœuds et des relations

Question 6

Afficher le nœud qui possède comme *nickname* The Imp.

Question 7

Affichez le clan qui a pour *clanname* The Starks.

Question 8

Affichez tous les personnages qui ont une relation *:LIER* avec le clan The Starks.

Indices :

```
1 match (c:clan {clanname:'The Starks'})<-[...]-(...) return ...
```

```
1 match (c:clan {clanname:'The Starks'})<-[l:...]->(p) return l,c,p
```

Question 9

Affichez toutes les relations de type *:TUE*, pour savoir qui a tué qui.

Indice :

```
1 match (:personnage)-[...]>(:personnage)
2 return result
```

Modélisation logique arborescente en XML



IV

A. Cours

1. Introduction à XML

a) Définition du XML

Définition : XML

L'eXtensible Markup Language XML [w3c.org/XML] est un **méta-langage** permettant de définir des langages à **balises**.

Il est standardisé comme une recommandation par le W3C depuis **1998**.

Exemple

- LaTeX et HTML sont des langages à balises (mais ce ne sont pas des langages XML)
- XHTML est un langage XML

Fondamental

En tant que méta-langage XML sert à définir des **formats** informatiques, c'est à dire des façons de représenter de l'information.

Les bonnes caractéristiques d'XML - **non-ambiguïté, lisibilité, passivité** - en font un très bon candidat pour de très nombreux usages, il est donc utilisé dans des secteurs très variés de l'informatique.

On distingue généralement deux grandes catégories d'utilisation : **les langages orientés données et les langages orientés documents**.

Complément : Versions de XML

La version historique de XML est la version 1.0, qui reste aujourd'hui la plus largement utilisée. Il existe également une version 1.1 de XML, qui propose des différences mineures et nécessaires uniquement dans des contextes particuliers. Cette nouvelle version a été nécessaire pour des raisons de compatibilité des outils existants. Les évolutions principales de XML 1.1 sont de nouveaux caractères permis pour les noms d'éléments (pour suivre l'évolution

d'Unicode depuis 1998), de nouvelles conventions pour les caractères de fin de ligne (pour la compatibilité avec des ordinateurs *main frame*) et de nouveaux caractères de contrôle dans le contenu.

Complément : Voir aussi

XML : Un langage à balise

XML : un méta-langage

Langages XML orientés données

Langages XML orientés documents

Caractéristiques recherchées pour un format XML

b) Document bien formé

Définition

Un document est dit bien formé lorsqu'il respecte les règles syntaxiques du XML.

Fondamental

XML ne pose pas de sémantique à priori mais uniquement des règles syntaxiques.

Syntaxe

Les règles syntaxiques à respecter sont :

- Il n'existe qu'un seul élément père par document, cet élément contenant tous les autres. Il est également appelé **racine**.
- Tout élément fils est inclus complètement dans son père (pas d'éléments croisés).

Attention

XML ne définit pas le comportement ni la manière dont doit être traité un document, mais uniquement un format.

Complément

Balise

c) Exemple : Un document XML

Exemple : Un mail

```

1  <?xml version='1.0' encoding='iso-8859-1'?>
2  <mail>
3      <de>stephane.crozat@utc.fr</de>
4      <a>fabrice.issac@utc.fr</a>
5      <objet>Demande d'information</objet>
6      <date>01-03-2001</date>
7      <corps>
8          <paragraphe>Bonjour Fabrice,</paragraphe>
9          <paragraphe>Peux tu m'expliquer ce qu'est le langage XML ?</paragraphe>
10         <paragraphe>Il me faudrait en effet ton avis éclairé sur le
11         sujet.</paragraphe>
12         <paragraphe>J'attends ta réponse, à bientôt</paragraphe>
13     </corps>
14 </mail>

```

d) Exercice

Compléter les trous avec les balises **fermantes** adéquates.

<?xml version="1.0"?>

```

<document>
  <entete>
    <titre>Document à corriger</titre>
    <auteur>Stéphane Crozat
    <date>01-03-01
    <version numero="1"/>

  <corps>
    <division titre="Première partie">
      <paragraphe>Ce texte doit être <important>corrigé</paragraphe>
      <paragraphe>Le contenu est sans importance ici</paragraphe>
    </division>
    <division titre="Seconde partie">
      <paragraphe>Ai-je le droit de mettre du texte ici ?</paragraphe>

```

e) Langages XML orientés données

Définition

Ils permettent d'enregistrer et de transporter des données informatiques structurées (comme par exemple des données gérées par des bases de données) selon des formats ouverts (c'est à dire dont on connaît la syntaxe) et faciles à manipuler (les structures arborescentes XML étant plus riches que des fichiers à plat par exemple).

Les langages XML orientés données servent surtout à l'échange ou la sérialisation des données des programmes informatiques.

Remarque

L'utilisation d'XML est en fait ici assez accessoire, d'autres formalismes s'y substituent sans difficulté, souvent moins explicites pour la manipulation humaine et souvent plus performants pour la manipulation informatique : CSV, JSON, ... (voir par exemple : <http://www.xul.fr/ajax-format-json.html>¹⁷ pour une comparaison JSON / XML).

Remarque : Format verbeux

XML est en général plus verbeux que ses alternatives (il contient plus de caractères), c'est pourquoi il est plus explicite pour un humain (ce qui n'est pas toujours utile), et moins performant informatiquement (ce qui n'est pas toujours un problème).

Exemple : Formats XML orientés données standards

- MathML, ChemML, ...
- ATOM, RSS
- Dublin Core
- RDF, OWL
- SVG
- ...

17 - <http://www.xul.fr/ajax-format-json.html>

Exemple : Formats XML orientés données locaux

XML est utilisé pour de nombreux langages orientés données locaux, en fait chaque fois qu'un format XML est défini pour les besoins spécifique d'un programme.

```

1 <myVector>
2   <x>5</x>
3   <y>19</y>
4 </myVector>

```

Complément : Langages XML de programmation

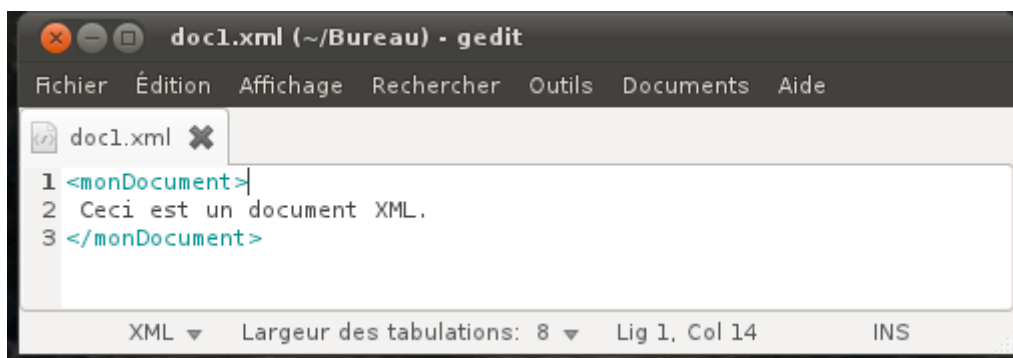
Le formalisme XML est également utilisé pour définir des langages de programmation, à l'instar du C ou du Java. Les langages de programmation écrits en XML sont généralement à vocation déclarative et adressent le plus souvent des traitements eux mêmes liés à XML.

XSL-XSLT est un exemple de langage de programmation écrit en XML. On peut également citer par exemple le langage de script ANT (<http://ant.apache.org/>¹⁸) ou XUL pour la réalisation d'IHM (<http://www.xul.fr/>¹⁹)

f) Outillage XML

Fondamental

Un éditeur de texte suffit pour faire du XML.



Document XML créé avec un simple éditeur de texte (gedit)

Parseurs

Un *parser* (ou parseur) XML est un programme capable d'analyser un document XML afin de :

- vérifier qu'il est bien formé
- éventuellement vérifier sa validité par rapport à un schéma (DTD, W3C Schema, RelaxNG)
- éventuellement en fournir une représentation mémoire permettant son traitement (SAX, DOM)

Par exemple Xerces est un parseur Java (<http://xerces.apache.org/xerces2-j/>²⁰). Tous les langages de programmation proposent aujourd'hui des parseurs XML.

Éditeur valideur

Un éditeur valideur est un éditeur spécialisé dans l'écriture du XML (on parle simplement d'éditeur XML), permettant de :

- vérifier dynamiquement que le fichier est bien formé,
- de charger des schémas pour vérifier dynamiquement la validité,

18 - <http://ant.apache.org/>

19 - <http://www.xul.fr/>

20 - <http://xerces.apache.org/xerces2-j/>

- de proposer des fonctions d'auto-complétion,
- ...

Éditeurs XML orientés développeurs

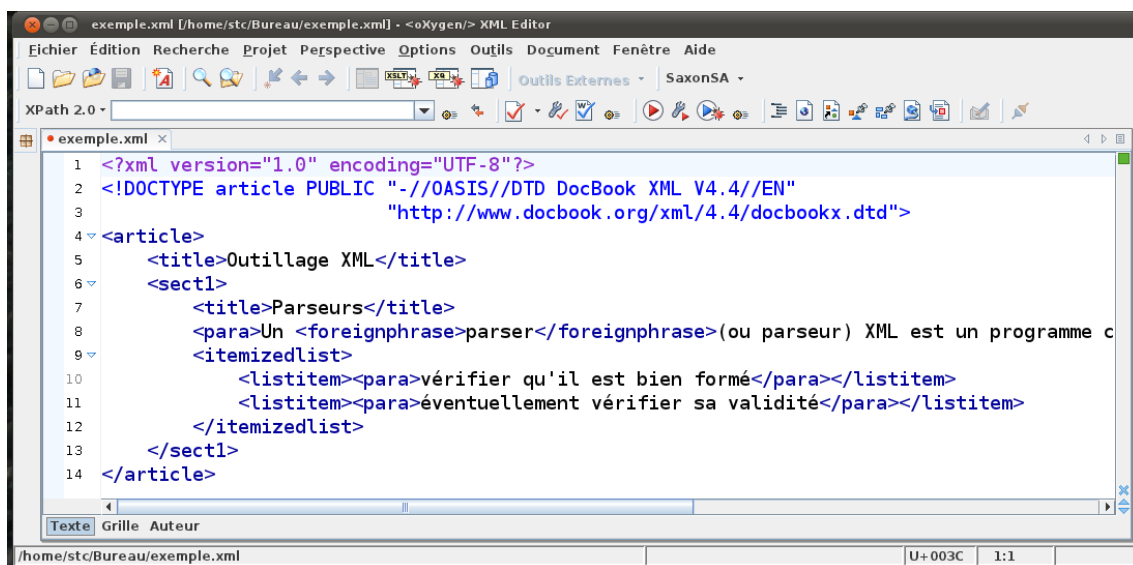
Les éditeurs spécialisés orientés développeurs sont des IDE permettant l'écriture :

- des schémas,
- des transformations,
- des fichiers XML de test,
- ...

Ils sont en général peu adaptés à la production des documents XML par les utilisateurs finaux (rédacteurs).

On peut citer par exemple :

- oXygen (<http://www.oxygenxml.com/>²¹), un produit commercial complet accessible à moins de 100€ pour les usages non commerciaux, multi-plateformes (Linux, Mac, Windows) ;
- XML Spy, disponible uniquement sous Windows
- ...



Exemple de fichier XML dans l'éditeur oXygen

Éditeurs XML orientés rédacteurs

Les éditeurs XML orientés rédacteurs prennent en général en entrée un schéma, des fichiers permettant de configurer l'éditeur, des programmes de transformation, et offre un environnement dédié à l'écriture et la publication de document XML.

On peut citer :

- Jaxe, libre et multi-plateformes (<http://jaxe.sourceforge.net/fr/>²²)
- Adobe Framemaker, sous Windows (<http://www.adobe.com/products/framemaker.html>²³)
- Arbortext, sous Windows (<http://www.ptc.com/products/arbortext/>²⁴)
- XMetal, sous Windows (<http://na.justsystems.com/content-xmetal>²⁵)

21 - <http://www.oxygenxml.com/>

22 - <http://jaxe.sourceforge.net/fr/>

23 - <http://www.adobe.com/products/framemaker.html>

24 - <http://www.ptc.com/products/arbortext/>

25 - <http://na.justsystems.com/content-xmetal>

- Scenari, chaîne éditoriale complète libre et multi-plateformes (<http://scenari.org/>²⁶)

Complément : Voir aussi

Définition des chaînes éditoriales XML

Définition de Scenari

2. Syntaxe de base XML

a) Balise

Définition : Balise

Les balises sont les composants fondamentaux permettant l'écriture de documents XML.

Elles se distinguent du contenu en utilisant les caractères <, > et /. Elles n'ont pas de présentation ou de signification définie par le langage mais elles auront un sens pour les applications et/ou le lecteur.

Syntaxe

Il existe trois types de balises :

- balise d'ouverture : <nom_balise>
- balise de fermeture : </nom_balise>
- balise vide : <nom_balise/>

Exemple : Exemple de balise XML

```
1 <adresse>12, rue de Paris</adresse>
```

b) Élément

Définition : Élément

Un élément XML est un extrait d'un fichier XML comprenant :

- une balise ouvrante
- une balise fermante
- le contenu compris entre les deux balises, qui peut être du **texte** et/ou d'autres **éléments**, ou **rien** (élément vide)

Le nom d'un élément peut être composé de tout caractère alphanumérique plus _, - et .. De plus, ils doivent commencer par un caractère alphabétique ou _ et ceux commençant par la chaîne xml sont réservés (que ce soit en minuscules, majuscules ou un mélange des deux).

Attention : Case-sensitive

XML différencie les majuscules des minuscules, il faut donc respecter la casse.

Attention

Deux éléments ne peuvent pas avoir un contenu croisé : <nom1> ... <nom2> ... </nom1> ... </nom2> est interdit.

Définition : Élément vide

On appelle élément vide un élément qui ne comporte rien entre sa balise ouvrante et sa balise fermante.

26 - <http://scenari.org/>

Syntaxe : Élément vide

Les syntaxes `<element></element>` et `<element/>` sont strictement équivalentes.

c) Attribut

Définition

Un attribut est une information supplémentaire attachée à un élément, on parle de métadonnée.

Syntaxe

Les attributs d'un élément sont formés d'une suite d'affectations séparées par des espaces : `attribut1='valeur' attribut2='valeur' ...`

Ils sont ajoutés à la balise ouvrante ou à une balise vide (jamais à une balise fermante) :

- `<nom_element [attributs]>`
- `<nom_element [attributs]/>`

La valeur est indiquée entre apostrophes ou guillemets (au choix, mais pas de mélange des deux) :

- `attribut1='valeur' OU`
- `attribut1="valeur"`

Méthode

Utilisez des apostrophes si la valeur de l'attribut inclut des guillemets et vice et versa.

Attention

Un élément ne peut pas contenir deux attributs ayant le même nom.

Syntaxe

Le nom d'un attribut est soumis aux mêmes contraintes que les noms d'éléments.

La valeur de l'attribut quant à elle peut contenir tout caractère à l'exception de ^, % et &.

Remarque : Équivalence attribut / élément

Un attribut peut toujours être représenté alternativement par un élément fils de l'élément qu'il caractérise, avec une signification du même ordre :

- `<element attribut="x"/>` et
- `<element><attribut>x</attribut></element>` sont similaires.

Il est donc tout à fait possible de faire du XML sans utiliser d'attribut.

Méthode : Usage des attributs

On utilise généralement les attributs :

- Pour différencier le contenu destiné à être affiché dans le document lisible des métadonnées qui ne le seront pas (version, date de création, ...)
- Pour simplifier l'écriture du document
- Pour ajouter des identifiants et des références

d) Structure générale d'un fichier XML

Syntaxe

Un document XML est constitué de :

- Un **prologue**

Il est facultatif et comprend une déclaration XML, indiquant la version du langage XML utilisé et le codage des caractères dans le document. Chacune de ces informations est optionnelle mais leur ordre est obligatoire

```
<?xml version="numéro de version" encoding="encodage des caractères"?>
```

- Un **arbre d'éléments** contenant au moins un élément (l'élément racine)

Exemple : Prologue

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

Indique que le document est codé en utilisant un langage XML de version 1.0, avec des caractères codés selon la norme UTF-8.

Exemple : Arbre d'éléments

```
1 <lettre>
2   <expediteur>moi</expediteur>
3 </lettre>
```

e) Exemple de fichier XML : un courriel

Exemple : Un courriel imprimé

Date: Mar, 28 Oct 2003 14:01:01 +0100 (CET)

De : Marcel <marcel@ici.fr>

A : Robert <robert@labas.fr>

Sujet: Hirondelle

Salut,

Pourrais-tu m'indiquer quelle est la vitesse de vol d'une hirondelle transportant une noix de coco ?

A très bientôt,

Marcel

Représentation possible de ce message en XML

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <email>
3   <entete>
4     <date type="JJMMAAAA">28102003</date>
5     <heure type="24" local="(GMT+01 :00)">14:01:01</heure>
6     <expediteur>
7       <adresse mail="marcel@ici.fr">Marcel</adresse>
8     </expediteur>
9     <recepteur>
10      <adresse mail="robert@labas.fr">Robert</adresse>
11    </recepteur>
12    <sujet>Hirondelle</sujet>
13  </entete>
14  <corps>
15    <salutation>Salut,</salutation>
16    <paragraphe>Pourrais-tu m'indiquer quelle est la vitesse de vol d'une
    hirondelle
17      transportant une noix de coco ?</paragraphe>
18    <politesse>A très bientôt,</politesse>
19    <signature>Marcel</signature>
20  </corps>
21 </email>
```

3. Introduction aux schémas XML

a) Notion de document valide

Définition : Schéma

Un schéma est une description de la structure que doit respecter un document lui faisant référence, c'est à dire qu'il établit la liste des éléments XML autorisés (avec leurs attributs), ainsi que l'agencement possible de ces éléments.

On parle aussi de **grammaire**, au sens où le schéma définit l'enchaînement autorisé des balises et vient en **complément de la syntaxe XML** (qui elle est indépendante d'un schéma particulier).

Définition : Document valide

Un document XML bien formé est dit valide pour un schéma donné s'il respecte les règles structurelles imposées par ce schéma.

Ce contrôle de la structure permet :

- De s'assurer l'homogénéité structurelle des documents de même type.
- Le traitement automatique d'un ensemble de documents de même type (mise en forme, stockage, extraction d'informations...).
- La création de formats standard et leur respect.

Exemple : Exemples de langages de schéma

Il existe plusieurs langages de définition schéma, mais les trois principaux sont :

- Document Type Définition (W3C) : Un langage hérité de SGML qui fait partie du standard XML
- W3C XML Schema (W3C) : Une alternative aux DTD destiné à moderniser et compléter ce langage historique
- Relax NG (OASIS, ISO) : Une autre alternative, compromis entre W3C XML Schema et DTD

b) Document Type Definition

Le formalisme de définition de schéma DTD est le premier qui a été introduit dès la première version du standard XML. Il est en fait intégré au standard W3C de XML.

Il est directement hérité de la norme SGML.

Les DTDs utilisent un langage spécifique (non XML) pour définir les règles structurelles. Un fichier de DTD peut contenir principalement deux types de déclarations :

- **des déclarations d'éléments,**
indiquent les éléments pouvant être inclus dans un document et l'organisation du contenu de chaque élément (éléments fils ou texte).
- **des déclarations d'attributs,**
définissent les attributs pouvant être associés à un élément ainsi que leur type.

Exemple : Exemple de DTD

```

1 <!ELEMENT document (paragraphe+)>
2 <!ATTLIST document type CDATA #REQUIRED>
3 <!ELEMENT paragraphe (#PCDATA)>

```

Exemple : Exemple de document XML valide

```

1 <?xml version='1.0' encoding='iso-8859-1'?>
2 <!DOCTYPE document SYSTEM "document.dtd">
3 <document type='memo'>
4   <paragraphe>Lorem ipsum dolor sit amet.</paragraphe>
5   <paragraphe>Consectetur adipiscing elit.</paragraphe>
6   <paragraphe>Sed do eiusmod tempor.</paragraphe>
7 </document>

```

c) Exercice

Le fichier *file.xml* n'est pas valide par rapport à la DTD *schema.dtd*. Sélectionnez les éléments causes de cette non-validité.

```

1 <?xml version="1.0"?>
2 <!--file.xml-->
3 <!DOCTYPE papier SYSTEM "schema.dtd">
4 <papier>
5   <titre>Réinterroger les structures documentaires</titre>
6   <auteur>Stéphane Crozat</auteur>
7   <auteur>Bruno Bachimont</auteur>
8   <resume>Nous proposons dans cet article d'aborder ...</resume>
9   <abstract>In this paper we define...</abstract>
10 </papier>

```

```

1 <!-- schema.dtd-->
2 <!ELEMENT papier (titre, sousTitre?, auteur, resume)>
3 <!ELEMENT titre (#PCDATA)>
4 <!ELEMENT sousTitre (#PCDATA)>
5 <!ELEMENT auteur (#PCDATA)>
6 <!ELEMENT resume (#PCDATA)>

```

- 1 - auteur
- 2 - abstract
- 3 - sousTitre
- 4 - titre
- 5 - resume

Éléments correctement spécifiés

Éléments incorrectement spécifiés

d) Relax NG : Syntaxe XML

Syntaxe : Syntaxe générale

```

1 <grammar xmlns="http://relaxng.org/ns/structure/1.0">
2   <start>
3     <element name="...">
4       ...
5     </element>
6   </start>
7 </grammar>

```

Syntaxe : Imbrication

```

1 <element name="">
2   <element name="">
3     ...
4   </element>

```

```
5 </element>
```

Syntaxe : Séquentialité

```
1 <element name="">
2   <element name=""> ... </element>
3   <element name=""> ... </element>
4   ...
5 </element>
```

Syntaxe : Attributs

```
1 <element name="">
2   <attribute name="">
3   ...
4 </attribute>
5 </element>
```

Syntaxe : Nœuds texte

```
1 <element name="">
2   <text/>
3 </element>
```

Syntaxe : Cardinalité

```
1 <element name="">
2   <zeroOrMore>
3     <element name=""> ... </element>
4   </zeroOrMore>
5   <oneOrMore>
6     <element name=""> ... </element>
7   </oneOrMore>
8   ...
9 </element>
```

Syntaxe : Optionalité

```
1 <element name="">
2   <optional>
3     <element name=""> ... </element>
4   </optional>
5   ...
6 </element>
```

Syntaxe : Alternative

```
1 <element name="">
2   <choice>
3     <element name=""> ... </element>
4     <element name=""> ... </element>
5     <element name=""> ... </element>
6   </choice>
7   ...
8 </element>
```

Syntaxe : Énumération

```
1 <attribute name="">
```

```

2      <choice>
3          <value>a</value>
4          <value>b</value>
5          <value>c</value>
6      </choice>
7  </attribute>

```

Complément

<http://www.oasis-open.org/committees/relax-ng/tutorial.html>²⁷

<http://xmlfr.org/oasis/committees/relax-ng/tutorial-20011203-fr.html>²⁸

e) Types de données

Syntaxe

Ajouter l'attribut `datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"` à la racine `grammar`.

```

1  <grammar
2      xmlns="http://relaxng.org/ns/structure/1.0"
3      datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
4      ...
5      <element name="...">
6          <data type="...">
7      </element>
8  </grammar>

```

Définition : Types primitifs

- string
- boolean
- decimal
- float, double
- date, dateTime, duration, time, gYearMonth, gYear, gMonthDay, gDay, gMonth
- hexBinary, base64Binary
- anyURI
- QName, NOTATION
- Types hérités des DTD : ID, IDREF, IDREFS...

Complément : Spécification des primitive datatypes

<http://www.w3.org/TR/xmlschema-2/>²⁹

Facette

Paramètre de spécialisation des types primitifs.

Exemple : Type string

Facettes :

- length : longueur de la chaîne
- pattern : expression régulière permettant de valider la chaîne par rapport au patron (définition en intention)
- enumeration : liste de valeurs autorisées (définition en extension)

27 - <http://www.oasis-open.org/committees/relax-ng/tutorial.html>

28 - <http://xmlfr.org/oasis/committees/relax-ng/tutorial-20011203-fr.html>

29 - <http://www.w3.org/TR/xmlschema-2/>

- ...

Définition : Built-in datatypes

Dérivés des types primitifs.

Par exemple :

- integer, dérivé de decimal
- normalizedString, dérivé de string
- language, dérivé de string
- ID, IDREF

Exemple : RelaxNG XML

```

1 <grammar xmlns="http://relaxng.org/ns/structure/1.0">
2   <start>
3     <element name="mail" datatypeLibrary="http://www.w3.org/2001/XMLSchema-
      datatypes">
4       <data type="string">
5         <param name="pattern">([ ^ ])+@([ ^ ])+.([ ^ ])+</param>
6       </data>
7     </element>
8   </start>
9 </grammar>

```

Exemple : RelaxNG compact

```

1 datatypes xsd="http://www.w3.org/2001/XMLSchema-datatypes"
2 start = element age {xsd:decimal {maxInclusive="100"}}

```

Complément

<http://www.xml.dvint.com/docs/SchemaDataTypesQR-2.pdf>³⁰

Complément

Guidelines for using W3C XML Schema Datatypes with RELAX NG :
<http://relaxng.org/xsd.html>³¹

f) Présentation de oXygen XML Editor

Oxygen XML Editor est un environnement de développement XML.

Il permet notamment de

- créer des documents XML bien formés,
- créer des schémas et valider des documents XML
- créer des transformations XSLT et les exécuter sur des documents XML

Toutes ces fonctions sont disponibles à partir du menu : Document > Document XML.

Il permet également de tester des expressions XPath sur un document XML ouvert dans l'éditeur.

Installation

Il peut être téléchargé ici : <http://www.oxygenxml.com>³² et il est possible de bénéficier d'une licence d'essai de 30 jours.

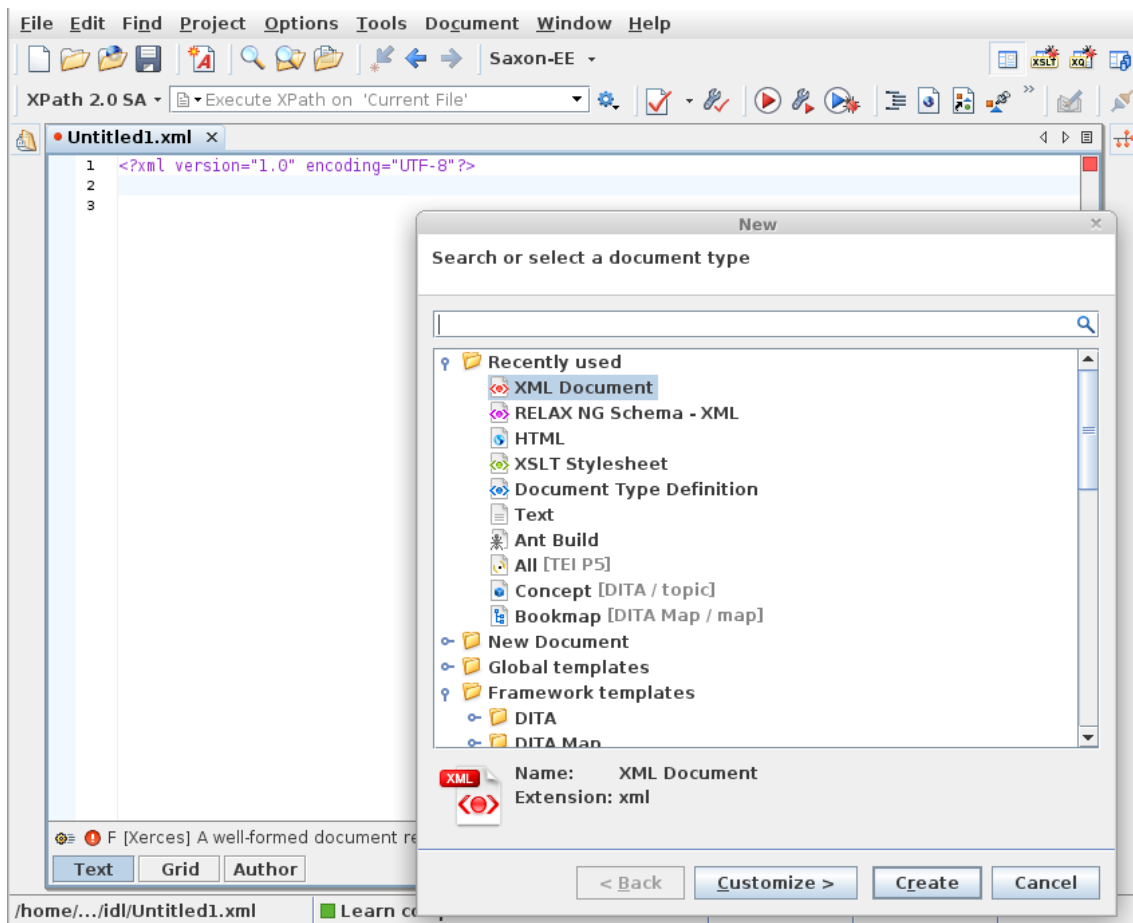
30 - <http://www.xml.dvint.com/docs/SchemaDataTypesQR-2.pdf>



31 - <http://relaxng.org/xsd.html>

32 - <http://www.oxygenxml.com/>

Les anciennes versions d'Oxygen peuvent être téléchargées à l'adresse : http://www.oxygenxml.com/software_archive_editor.html³³

Interface



- Menu **File** pour créer des fichiers XML, des schémas (DTD, RNG, XSD) ou des transformations XSLT.
- Menu **Document** pour vérifier qu'un document est bien formé, le valider, le transformer.
- Icône  pour valider ou vérifier que le document est bien formé.
- Icône  pour exécuter les transformations.
- Zone de saisie et test des XPath :

XPath 2.0 SA ▾ ▾ Execute XPath on 'Current File' ⚙

Complément

Fichiers XML

Schémas XML

Transformations XSLT

33 - http://www.oxygenxml.com/software_archive_editor.html

4. Manipulation XML avec XPath

a) L'arbre du document XML

Il est possible de représenter un document XML sous forme d'arbre, tel que :

- L'arbre possède une racine / qui a comme fils l'élément racine
- l'élément racine est l'élément du document XML qui contient tous les autres
- chaque nœud a comme fils les éléments et le texte qu'il contient, ainsi que ses attributs.

Exemple : Fichier XML

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <document modele="ULCoursGeneral" code="BP-Incendie3_S1_E2_UL1">
3   <entete>
4     <identification>
5       <titre>L'assurance de la responsabilité de voisinage</titre>
6       <date>21/02/01</date>
7       <auteur>AEA</auteur>
8       <version>1.00</version>
9     </identification>
10  </entete>
11  <corps>
12    <contenu>
13      <paragraphe>Cette garantie est appelée : recours des voisins et des
14      tiers.</paragraphe>
15      <remarque>
16        <paragraphe>L'image suivante <ressource URIsrc="img07.jpg"
17        titre="Recours des voisins et des tiers" type="image"/>
18        montre la
19          garantie.</paragraphe>
20      </remarque>
21    </contenu>
22  </corps>
23 </document>

```

Exemple : Arbre XML

```

1 /
2 |document
3   |@modele = "ULCoursGeneral"
4   |@code = "BP-Incendie3_S1_E2_UL1"
5   |entete
6     |identification
7       |titre
8         |text() = "L'assurance de ..."
9       |date
10        |text() = "21/02/01"
11      |auteur
12        |text() = "AEA"
13      |version
14        |text() = "1.00"
15    |corps
16      |contenu
17        |paragraphe
18          |text() = "Cette garantie ..."
19        |remarque
20          |paragraphe
21            |text() = "L'image suivante"
22            |ressource
23              |@URIsrc = "img07.jpg"
24              |@titre = "Recours des voisins..."
25              |@type = "image"
26            |text() = "montre la garantie."

```

Remarque : Ordre des nœuds

L'ensemble des nœuds de l'arbre d'un document est muni d'un ordre, qui est celui de l'ordre dans le document XML sérialisé.

b) Introduction à XPath

Définition : Expression XPath

XPath est un langage d'expressions permettant de pointer sur n'importe quel élément d'un arbre XML depuis n'importe quel autre élément de l'arbre.

- Une expression XPath peut-être **absolue** (sa résolution est **indépendante** d'un contexte ou nœud courant : elle commence dans ce cas par `/`).
- Une expression XPath peut-être **relative** (sa résolution est **dépendante** d'un contexte ou nœud courant : elle ne commence dans ce cas pas par `/`, elle peut commencer par `./` (syntaxe développée)).

Fondamental

Une expression XPath renvoie

- un **node-set**, ou ensemble de nœuds, c'est à dire un sous-arbre de l'arbre du document
- une chaîne de caractères
- un booléen
- un réel

Exemple : Exemples d'expressions XPath

1	/document/entete/identification/titre
2	/document/@modele
3	corps//contenu
4	contenu/*
5	contenu/remarque[1]
6	../paragraphe
7	@type

Complément : Types de nœuds XPath

- root nodes
- element nodes
- text nodes
- attribute nodes
- namespace nodes
- processing instruction nodes
- comment nodes

<http://www.w3.org/TR/xpath/#data-model>³⁴

Complément

Pour une introduction à XPath : Brillant07 [Brillant07] pp.123-129

Complément : Voir aussi

L'arbre du document XML

Syntaxe XPath

34 - <http://www.w3.org/TR/xpath/#data-model>

c) Exercice

Soit le fichier XML ci-après.

Le nœud courant est un des éléments *terme*, écrivez 4 expressions XPath différentes permettant de renvoyer le titre du document :

1. Sachant que *titre* est unique dans tout le document.
2. Sachant que *titre* est le fils de l'élément racine *papier*.
3. Sachant que *titre* est le fils du père du père du nœud courant.
4. Sachant que *titre* est avant le nœud courant dans l'ordre du document.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <papier type="scientifique">
3      <titre>Réinterroger les structures documentaires</titre>
4      <sousTitre>De la numérisation à l'informatisation</sousTitre>
5      <auteur>Stéphane Crozat</auteur>
6      <auteur>Bruno Bachimont</auteur>
7      <resume>Nous proposons dans cet article d'aborder ...</resume>
8      <abstract>In this paper we define...</abstract>
9      <motsCles>
10         <terme>Ingénierie des connaissances</terme>
11         <terme>XML</terme>
12         <terme>Document</terme>
13     </motsCles>
14     <keywords>
15         <word>Knowledge engineering</word>
16         <word>XML</word>
17         <word>Document</word>
18     </keywords>
19     <publication date="2004-07-05"/>
20     <version maj="1" min="0"/>
21     <ressource
22         uriSrc="http://archivesic.ccsd.cnrs.fr/docs/00/06/23/97/PDF/sic_00001015.pdf"/>
    </papier>

```

1. //
2. /
3. ..
4. preceding

B. Exercice

1. Mon nom est personne

Modélisation XML

45 min

Question 1

Écrivez un document XML bien formé permettant de représenter des personnes, avec leur nom, leur prénom et leur âge.

On représentera deux personnes.

Indice :

Document bien formé

Balise

Exemple : Un document XML

Question 2

Écrivez un schéma au format DTD permettant de valider le document précédent, sachant que le nom et le prénom sont obligatoires, mais pas l'age.

Indice :

Notion de document valide
Document Type Definition
DTD : Déclaration d'éléments

Question 3

Écrivez à nouveau le schéma, mais au formalisme RelaxNG, en utilisant le **typage des données** pour représenter l'age comme un *integer*.

Indice :

Relax NG : Syntaxe XML
RelaxNG : Types de données

Question 4

Écrivez des expressions XPath permettant de sélectionner :

- les noms des personnes, c'est à dire les éléments `nom` qui appartiennent à des éléments `personne`, qui appartiennent à l'élément `personnes` qui est à la racine.
- le nom de la seconde personne
- les noms des personnes dont l'age est renseigné
- les noms des personnes qui ont plus de 30 ans

Indice :

Introduction à XPath

Question 5

Testez vos propositions avec un éditeur XML validant.

Indice :

Présentation de oXygen XML Editor

2. Glossaire I

Soit le fichier XML suivant permettant de représenter un glossaire.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <glossaire>
3   <definition id="xml">
4     <terme>XML</terme>
5     <explication>XML est un méta-langage.</explication>
6   </definition>
7   <definition id="sgml">
8     <terme>SGML</terme>
9     <explication>SGML est un méta-langage.</explication>
10    <voirAussi ref="xml"/>
11  </definition>
12 </glossaire>

```

Pour les cinq questions suivantes, le nœud courant est `glossaire`.

Question 1

Que renvoie l'expression XPath suivante : `./*`

Question 2

Que renvoie l'expression XPath suivante : `./explication`

Question 3

Que renvoie l'expression XPath suivante : `//terme/text()`

Question 4

Écrire le XPath permettant de renvoyer les nœuds `terme`, qui ont `definition` comme père.

Question 5

Écrire le XPath permettant de renvoyer les nœuds `voirAussi` qui ont la valeur `xml` pour leur attribut `ref`.

Pour les trois questions suivantes, le nœud courant est le premier nœud `definition`, celui dont l'attribut `id="xml"`.

Question 6

Écrire le XPath permettant de renvoyer les nœuds `voirAussi` qui ont la valeur `xml` pour leur attribut `ref`.

Question 7

Écrire le XPath permettant de renvoyer les nœuds `definition` qui contiennent un élément `voirAussi` qui a la valeur `xml` pour son attribut `ref`.

Question 8

Écrire le XPath permettant de renvoyer le texte des nœuds `terme` fils des `definition` qui contiennent un élément `voirAussi` qui a la valeur `xml` pour son attribut `ref`.

Pour les deux questions suivantes, le nœud courant est un nœud `definition` quelconque.

Question 9

Écrire le XPath permettant de tester si le nœud courant contient un élément `voirAussi` qui se référence lui même (qui a la même valeur pour `ref` que l'attribut `id`).

Question 10

Écrire le XPath permettant de renvoyer les nœuds suivants qui se référencent eux-mêmes.

C. Devoir

1. On l'appelle Trinita

Soit la table `Personne` (`#id:integer`, `nom:varchar`, `prenom:varchar`, `age:integer`) avec `nom NOT NULL`, `prenom NOT NULL`.

Question 1

Créez la base de données relationnelle correspondante. Instanciez-là avec deux enregistrements.

Soit le fichier XML suivant :

1	<?xml version="1.0" encoding="UTF-8"?>
2	<personnes>
3	<personne>
4	<nom>Personne</nom>
5	<prenom>Terence</prenom>
6	</personne>

```

7      <personne>
8          <nom>Trinita</nom>
9          <prenom>Terence</prenom>
10         <age>40</age>
11     </personne>
12 </personnes>

```

Question 2

Créer la vue SQL permettant de préparer la création de ce format XML en générant pour chaque enregistrement une ligne `<personne>`.

Indices :

```

1      <personne><nom>Personne</nom><prenom>Terence</prenom></personne>
2
      <personne><nom>Trinita</nom><prenom>Terence</prenom><age>40</age></perso
      nne>

```

On utilisera l'opérateur `||` : `SELECT x || ' ' || y FROM t.`