

M1 – DAC

MLBDA – 4I801

Modèles et Langages pour les Bases de Données Avancées

Support de TD/TME

Site : <http://dac.lip6.fr/master/enseignement/ues/mlbda/>

Sommaire

SQL

Conception	4
SQL3	6
Syntaxe SQL3	10
XML DTD	15
XSchema	20
XPath	31
XQuery	34
Sparql	37

TD1 – SQL avancé

Exercice 1 : Requêtes avancées sur la base MONDIAL

Le schéma relationnel de la base MONDIAL est :

Continent (Name, Area)

Country (Name, Code, Capital, Province, Area, Population) // Province est la région de la capitale

Province (Name; Country, Population, Area, Capital, CapProv) // CapProv est la province de la capitale. Peut différer du nom de la province lorsque deux provinces «concentriques» ont la même capitale.

City (Name, Country, Province, Population, Longitude, Latitude)

Encompasses (Country, continent, percentage)

Borders (Country1, Country2, length) // cette relation est asymétrique. On a (F, I, ...) pour France-Italie et (E,F,...) pour Espagne-France. Country1 < Country2 selon l'ordre lexicographique.

Organization (Abbreviation, Name, City, Country, Province, Established) // Established est une date

IsMember(Country, Organization, type)

Population (Country, Population_growth, infant_mortality)

Economy (Country, GDP, Agriculture, Service, Industry, Inflation) // GDP=PIB

Politics (Country, Independence, Dependent, Government)

// date d'indépendance, dependent = code du pays d'attache

Language (Country, Name, Percentage)

EthnicGroup (Country, Name, Percentage)

Religion (Country, Name, Percentage)

Sea (Name, Depth)

Lake (Name, Area, Depth, Altitude, Type, River, Coordinates)

Island (Name, Islands, Area, Height, Type, Coordinates)

IslandIn (Island, Sea, Lake, River)

River (Name, River, Lake, Sea, Length, Source, Moutains, SourceAltitude, Estuary)

//le fleuve Name se jette dans un fleuve OU dans un lac OU dans une mer.

MergesWith (sea1, Sea2) // cette relation est asymétrique.

Desert (Name, Area, Coordinates)

Mountain (Name, Mountains, Height, Type, Coordinates)

MountainOnIsland (Mountain, Island)

Located (City, Province, Country, River, Lake, Sea) // ville en bordure de lac, mer et/ou fleuve

LocatedOn (City, Province, Country, Island) // ville sur une ile

Geo_Desert (Desert, Country, Province)

Geo_Estuary (River, Country, Province)

Geo_River (River, Country, Province)

Geo_Source (River, Country, Province)

Geo_Sea (Sea, Country, Province)

Geo_Island (Island, Country, Province)

Geo_Lake (Lake, Country, Province)

Geo_Mountain (Mountain, Country, Province)

Ecrire en SQL les requêtes suivantes

1. Le nom des pays membres des nations unies trié par nom de pays
2. Idem avec la population, trié décroissant par population
3. Le nom des pays NON membre des nations unies
4. Les pays frontaliers de la France (solution avec union)
5. Les pays frontaliers de la France (solution avec OR)
6. La longueur de la frontière française
7. Pour chaque pays, le nombre de voisins
8. Pour chaque pays, la population totale de ses voisins
9. Pour chaque pays d'Europe, la population totale de ses voisins
10. Les organisations, avec le nombre de membres et pop totale.
11. Les organisations regroupant plus de 100 pays, avec le nombre de membres et pop totale
12. Les pays d'Amérique avec leur plus haute montagne
13. (*) Les affluents directs du Nil : tous les fleuves qui se jettent dans le Nil.
14. (*) Tous les affluents du Nil : ceux qui s'écoulent directement ou indirectement dans le Nil.
15. (*) La longueur totale des cours d'eau alimentant le Nil, Nil inclus.
16. a) La plus grande organisation en termes de nombre pays membre
b) (*) Les 3 plus grandes organisations en termes de nombre pays membre
17. (*) La densité de population (exprimée en nombre d'habitants par km²) de la zone formée de l'Algérie et la Libye ainsi que de tous leurs voisins directs.
18. (*) Idem mais en enlevant tous les déserts de la zone en question.
19. Le pourcentage de croyants de chaque religion dans la population mondiale
20. Les couples de pays européens ayant exactement accès aux mêmes mers

TME 1 : Requêtes avancées sur la base MONDIAL

Lire les instructions sur le site web MLBDA, rubrique : *Les travaux dirigés et TME*

TD2

Conception objet et Schéma objet-relationnel

Exercice 1. Bars à bières : Schéma

L'application qu'on souhaite modéliser concerne une étude sur la consommation et la vente de bières dans les différentes villes de France. Dans ce but, on veut représenter et gérer les données concernant les bières et les activités liées à leur consommation et à leur vente dans divers établissements.

Une ville est désignée par un nom (identifiant) et un nombre d'habitants, et peut contenir plusieurs établissements de bières. Chaque établissement est désigné par un nom (identifiant) et un horaire d'ouverture, et propose une sélection de bières. On distingue deux types d'établissements: les *magasins*, et les *bars*.

Les *magasins* proposent des bières à l'achat, et sont caractérisés par une surface.

Les bars proposent des bières à la consommation (sur place), et disposent d'une certaine capacité d'accueil.

Chaque bière est identifiée par une marque et un type, et est caractérisée par un prix, un fabricant, un pays de fabrication, et est consommée par un certain nombre de consommateurs. Un consommateur est caractérisé par un id (identifiant), un nom, un âge, une ville de résidence, et peut consommer une variété de bières en les achetant dans différents magasins ou en les consommant sur place dans différents bars.

Question 1 Proposer un diagramme de cette application en utilisant la représentation graphique de la norme ODMG. Déterminer les racines de persistances correspondantes.

Question 3. On veut définir en SQL3 le sous-schéma suivant de l'application :

Un consommateur fréquente un ou plusieurs bars et consomme une ou plusieurs bières.

Une bière est consommée par un ou plusieurs consommateurs.

Proposer un schéma SQL3:

Question 4. (1pt) Créer les tables LesBars, LesBières, LesConsommateurs nécessaires au stockage des objets Bar, Bière, et Consommateur.

Exercice 2: Inscription pédagogique : Schéma

Question 1 : Définir un schéma SQL3 pour le cas suivant. Des étudiants s'inscrivent à l'université pour une certaine année (de 1 pour la Licence1 à 5 pour le Master2). On connaît leur numéro, leur nom, leur âge. Lors de l'inscription pédagogique, ils fixent leur contrat composé d'unités d'enseignement (UE). Une unité d'enseignement a un nom, un code et un nombre de crédits (3 ou 6). Elle est composée de plusieurs séances numérotées 1, 2, ... On connaît le sujet d'une séance et les étudiants présents à cette séance. La présence n'étant pas obligatoire, un étudiant peut être absent à certaines séances des unités d'enseignement de son contrat. Un étudiant a une note pour chaque unité d'enseignement de son contrat.

Question 2 : Stockage. Définir les tables pour stocker les objets de la base

Exercice 3. Avis sur les restaurants : Schéma

On considère le schéma SQL3 suivant (lire la 1ère colonne puis la 2ème).

1) Définir le schéma SQL3 pour un site gérant les avis des utilisateurs sur des restaurants. Un restaurant a un nom et un nombre d'étoiles, il est dans une ville. Une personne a un nom, des amis. Elle met des avis sur des restaurants. Un avis est soumis à une certaine date (année).

2) Si on ajoutait un attribut *habitués* dans Restau pour connaître qui sont les habitués fréquentant le restau avec pour chacun le nombre de fois où il est déjà venu. Quel serait le type de cet attribut ? Compléter le schéma. Rmq : dans les questions suivantes, on ignorera l'attribut *habitués*.

3) On considère seulement le schéma de l'énoncé (on ignore la question précédente). Compléter la définition des tables LesR et LesP stockant les restaurants et les personnes respectivement.

Exercice 4. Données géographiques pour le tourisme

L'objectif de cet exercice est d'étudier et de modéliser une base de données géographique en utilisant le standard ODMG. Cette base de données sera utilisée dans le cadre d'une application touristique.

I. Données

Les données concernent le territoire français, où on s'intéresse plus particulièrement aux régions et aux départements.

Une région est caractérisée par un nom (identifiant), une description, un plan, une préfecture, et un certain ensemble de départements.

Un département est caractérisé par un numéro (identifiant), un nom, une préfecture, une surface, et un ensemble d'agglomérations.

Une agglomération est identifiée par un numéro de code postal, elle possède un nom, un nombre d'habitants, une description (historique, géographique,...) et peut posséder certaines caractéristiques culturelles (monuments, spécialités culinaires, ...).

On distingue deux types d'agglomérations, les villes et les villages :

- Une ville est une agglomération possédant au moins 5000 habitants, et dispose d'une capacité d'accueil (restaurants, hôtel, hôtel-restaurant).
 - Un hôtel est caractérisé par un nom, une adresse, un nombre de lits et un nombre d'étoiles (de 0 à 5 étoiles).
 - Un restaurant est caractérisé par un nom, une adresse, un nombre de couverts, et un nombre de toques (de 0 à 5 toques).
- Un village est une agglomération de moins de 5000 habitants. On ne s'intéresse pas dans ce cas à la capacité d'accueil.

Les agglomérations sont reliées entre elles par un ou plusieurs réseaux de transport. On peut ainsi se déplacer d'une agglomération à une autre par le train (ter ou tgv), la route (RN à 2 voies ou autoroute à 4 voies) ou l'avion (hélice ou jet).

II. Traitements

L'application qui sera développée devra permettre l'exécution d'un certain nombre de requêtes administratives ou topologiques.

- Des requêtes administratives concernant les régions, les départements et les agglomérations :
 - « Quels sont les départements d'une région données ? »
 - « Quels sont les villes, les villages d'un département ou d'une région données ? »
 - « Superficie, nombre d'habitants d'un département ou d'une région ? »
- Des requêtes topologiques :
 - « Quels sont les départements **limitrophes** d'un département donné ? »
 - « Quels sont les trajets possibles entre deux agglomérations ? »
 - « Existe-t-il une liaison aérienne entre deux agglomérations données ? »
 - ...
- Des requêtes touristiques :
 - « Quelles sont les agglomérations possédant un intérêt touristique (culturel) ? »
 - « Quelles sont les agglomérations possédant des capacités d'accueil dans un département ? »
 - « Quelles sont les agglomérations possédant des hôtels trois étoiles ? »

Question. Représenter graphiquement les types abstraits et les différents liens qui les relie (héritage, association) en utilisant la représentation graphique de l'ODMG.

1. Quelles sont les relations (ou collections d'objets) nécessaires pour poser les requêtes R1 à R7 ?
2. Définir les relations permettant de stocker toutes les pièces de l'application dans la base de données (fichier instance.sql).
3. Ecrire les ordres de manipulation de données pour créer 3 matières : bois, fer, ferrite (fichier instance.sql),
4. Créer la procédure `insere_pieces_de_base` pour instancier les pièces de base 3 pièces en bois, 2 en fer et 1 en ferrite.
5. Créer la procédure `insere_pieces_composites` pour instancier les pièces composites.

Question 4 : Requêtes

Écrire les requêtes suivantes en SQL3. Pour chaque requête, donner le **type du résultat**.

- R1. Quel est le nom et le prix au kilo des matières ?
- R2. Quel est le nom des matières dont le prix au kilo est inférieur à 5 euros ?
- R3. Quelles sont les pièces de base en bois ?
- R4. Quel est le nom des matières dont le libellé contient 'fer'. Utiliser la syntaxe : `like '%fer%'`
- R5. Donner le nom des pièces formant la pièce nommée 'billard'. Le résultat est la liste {boule, canne, table}
- R6. Donner le nom de chaque matière avec son nombre de pièces de bases.
- R7. Quelles sont les matières pour lesquelles il existe au moins 3 pièces de base ?

Question 5 : Définition des méthodes

Compléter les types de données `Piece_base` et `Piece_Composite` avec des méthodes écrites en PL/SQL.

1. Écrire trois méthodes calculant respectivement le *volume*, la *masse* et le *prix* d'une pièce de base.
2. Écrire une méthode `nb_pieces_base` calculant le nombre de pièces de base entrant dans la fabrication d'une pièce composite.
3. Écrire une méthode `composee_de` donnant le **nom** des pièces de base contenues dans une pièce composite.
4. Écrire les méthodes calculant la *masse* et le *prix* d'une pièce composite.
5. (facultatif) Écrire une méthode `contenue_dans` donnant les pièces finales (*i.e.* les pièces principales n'entrant dans aucune pièce composite) contenant une pièce donnée.
6. (facultatif) Écrire les méthodes `init_piece_base` et `init_piece_composite` pour initialiser une pièce de base et une pièce composite.

Question 6 : Requêtes avec appel de méthode

Écrire les requêtes suivantes en SQL3. Pour chaque requête, donner le **type du résultat**.

- RM 1. Quels sont le nom, le volume, la masse et le prix des pièces de base ?
- RM 2. Quels sont le nom et la masse des pièces composites dont la masse est supérieure à 100 ?
- RM 3. Quelles sont les pièces composites contenant plus de 10 pièces ?
- RM 4. Quel est le nom des composants de chaque pièce composite ?
- RM 5. Quel est l'ensemble des pièces répertoriées dans la base ?
- RM 6. Quel est le nom des pièces composites qui ont une canne ?
- RM 7. Quel est le nom des pièces composites qui n'ont aucun clou ?
- RM 8. (*) Quel est le nom des pièces finales (*i.e.* celle qui n'entrent dans la fabrication d'aucune pièce) ?
- RM 9. (*) Combien de pièces de base entrent dans la composition d'une pièce donnée ?

Note : les questions marquées avec une (*) sont facultatives.

Exercice 2. Bars à bières : Insertions

On considère le schéma SQL3 de l'exercice Bars à bières du TD précédent.

La base contient deux bars : Le 'Bar du coin', d'une capacité de 100, et le bar 'Le Select', d'une capacité de 80. Elle contient également deux bières dont les marques sont 'Chimay' et 'Stella'.

- Ecrivez l'instruction SQL3 qui permet d'insérer le consommateur s'appelant 'Max', qui a l'identificateur 1, et qui a 22 ans. Il fréquente le bar 'Le Select' et consomme de la bière 'Stella'.
- Max consomme aussi de la bière 'Chimay'. Ecrivez une instruction SQL3 permettant d'insérer cette information dans la base.

Exercice 3: Inscription pédagogique : Insertion, Requêtes, Méthodes

Suite de l'exercice du TD précédent

Question 1 : Instanciation

- Insérer l'UE MLBDA code 4I801 pour 6 crédits. Les séances de cette UE sont dans l'ordre SQL, SQL3, XML.
- Insérer l'UE PLDAC 4I101 pour 6 crédits et avec les séances Intro et Méthodologie.
- Ajouter Alice 28 ans inscrite en Master1 avec un contrat ayant les UE 4I801 et 4I101.
- Enregistrer la présence d'Alice à la 3^{ème} séance de MLDBA.

Question 2 : Requêtes. Formuler les requêtes suivantes.

- Quels sont les étudiants âgés de plus de 25 ans et inscrits à l'unité de code 4I801 ? Le résultat affiche des objets Etudiant.
- Quel est le nom des étudiants présents à la séance 3 de l'unité nommée MLBDA ? Répondre en interrogeant la table LesU.
- Pour chaque étudiant inscrit à l'unité de code 4I801, afficher son nom et le nombre de séances qu'il a suivies. Le résultat contient aussi le nom des étudiants inscrits à 4I801 mais qui n'ont suivi *aucune* séance.
- Pour les étudiants présents à la 2^{ème} séance de MLBDA, afficher leur nom et le nom des unités auxquelles ils sont inscrits. Chaque ligne du résultat est un couple nomE, nomU.
- Pour chaque séance de chaque unité d'enseignement, quel est le nombre de **présents**? Afficher le code de l'unité, le numéro de la séance et le nombre de présents.
- Pour chaque étudiant et chaque unité à laquelle il est inscrit, quel est le nombre de séances ?
- Afficher le nom des étudiants présents à toutes les séances de toutes les unités de leur contrat ?

Question 3 : Méthodes

On considère la méthode créditTotal() du type Etudiant, retournant la somme des crédits des unités d'enseignement pour lesquelles l'étudiant a une note supérieure ou égale à 10 (en supposant qu'on connaît les notes d'un étudiant).

- Afficher le nom et le crédit total des étudiants qui ont un crédit total supérieur à 600.
- On considère la méthode nbUCommunes qui renvoie le nombre d'unités d'enseignement communes entre 2 étudiants. En invoquant cette méthode, écrire la requête qui affiche les paires d'objets étudiants qui ont 10 unités en commun. La signature de nbUCommunes est .

Member function nbUCommune(Etudiant e) return number ;

c) On complète le type `Etudiant` avec la méthode `notes()` qui retourne l'ensemble des couples (codeU, note) de l'étudiant, et dont la signature est :

```
member function notes return EnsC1;
```

en supposant que les types `EnsC1` et `C1` ont préalablement été définis comme suit :

```
create type C1 as object (code varchar2(30), note Number);      -- un couple (code unité, note)
/
create type EnsC1 as table of C1; -- un ensemble de couples
/
```

L'étudiant 1234 est inscrit à 2 unités. Il a eu 18 en 4I801 et 16 en 4I200. Quel est le résultat de la requête : `select e.notes() from LesE e where e.numéro='1234'` ;

Répondre en précisant tous les constructeurs des objets (et des ensembles d'objets) affichés.

En invoquant la méthode `notes()`, écrire la requête : quel est le code des unités d'enseignement où l'étudiant numéro 3456 a une note inférieure à 10 ? Ne pas afficher les notes mais seulement le code des unités.

d) On connaît les notes des étudiants inscrits à cette unité :

Ecrire le corps de la méthode `notes` (définie dans la question précédente) du type `Etudiant`.

Question 4 : Récursion

On complète le type `Unité` avec un attribut `prerequis` pour décrire les pré-requis d'une UE. Une UE peut avoir pour prérequis une ou plusieurs autres UE. On suppose que les pré-requis sont des UE du niveau précédent : une UE de M2 n'a que des pré-requis de M1, une UE de M1 n'a que des pré-requis de L3, ainsi de suite. Une unité de L1 n'a aucun pré-requis. On appelle `EnsU` le type du nouvel attribut `prerequis`.

a) Définir le type `EnsU`.

b) Pour le type `Unité`, définir la méthode récursive `prerequisTousNiveaux()` retourne l'ensemble des pré-requis directs et indirects d'une unité d'enseignement. Répondre un donnant la signature et le corps de la méthode `prerequisTousNiveaux()`.

SGBD objet relationnel : Langage SQL3

Syntaxe SQL3

La syntaxe simplifiée du langage SQL3 est exprimée au moyen d'une grammaire BNF. La syntaxe de la grammaire est :

un caractère du langage est représenté en **gras**, <mot> représente un élément non terminal

Les trois opérateurs d'occurrence sont :

(x)* représente 0 ou plusieurs fois l'élément x,

(x)+ représente 1 ou plusieurs fois l'élément x,

[x] représente une occurrence optionnelle de x.

x | y représente l'élément x OU y.

1. Définition du schéma

1.1 Définition d'un type de données

Compiler chaque définition de type **individuellement** avec la commande « / » ou @compile

<schema> = (<définition_type_objet> | <définition_type_ensembliste> | <déclaration_type>)*

<définition_type_objet> =

```
create type <type_objet> as Object (
  <nom_attr> [ref] <type>, )+
  <declaration_methodes>, )*
);
/
```

<définition_type_ensembliste> =

```
create type <type_ensembliste> as (Table | Varray(<longueur>)) of <type>;
/
```

Déclaration d'un type incomplet qui sera complété ultérieurement (utile pour définir un type mentionnant un autre type qui n'est pas encore défini).

```
<déclaration_type > = create type <type_objet>; --
/
```

<type> = <type_atomique> | <type_objet> | <type_ensembliste>

<type_atomique> = **Varchar2**(<longueur>) | **Number**(<longueur> [, <longueur>]) | **Date** | ...

1.2 Définition d'une association entre types

Une association est représentée en ajoutant des attributs aux types reliés par l'association. Lorsque l'arité (*i.e.* la cardinalité) de l'association est supérieure à 1, le type de l'attribut ajouté est ensembliste (table ou varray).

Le graphe des associations entre types doit être acyclique. Utiliser le type REF pour éviter les cycles.

a) Association 1-1 entre deux types X et Y :

Direction $X \rightarrow Y$:

Si l'association est une **agrégation** : `create type X as object (a Y, ...);`

sinon, `create type X as object (a ref Y, ...);`

+ contrainte d'unicité pour indiquer qu'il n'existe pas 2 objets X qui font référence au même objet Y.

Direction $Y \rightarrow X$: Solution symétrique.

b) Association 1-N entre deux types Y et X :

Si l'association est une **agrégation** :

`create type Y as object (...);` -- définir le type Y

`create type Ens_Y as varray(n) of Y;` -- définir le type "ensemble de Y" en utilisant *varray* ou *table of*

...

`create type X as object (a Ens_Y, ...);` -- définir le type X

sinon

Direction $X \rightarrow Y$ (X est associé à plusieurs Y)

`create type Y;` /

`create type Ens_Y as varray(n) of ref Y;` -- ou *table of ...*

`create type X as object (a Ens_Y, ...);`

Direction $Y \rightarrow X$ (Y est associé à un X)

`create type Y as object (b ref X, ...);`

c) Association N-M entre deux types X et Y :

Direction $X \rightarrow Y$

`create type Y;`

`create type Ens_Y as varray(n) of ref Y;`

`create type X as object (a Ens_Y, ...);`

Direction $Y \rightarrow X$

`create type Ens_X as varray(n) of ref X;`

`create type Y as object (b Ens_X, ...);`

1.3 Définition d'une méthode

<déclaration_méthode> = <déclaration_fonction> | <déclaration_procédure>

<déclaration_fonction> =

member function <nom_fonction> [(<nom_paramètre> in <type>, ...)]

return <type_du_resultat>

<déclaration_procédure> =

member procedure <nom_procédure> [(<nom_paramètre> in <type>, ...)]

Le corps des méthodes est défini ensuite au moyen de la commande :

create or replace type body <type_objet> **as**

```

<déclaration_methode> is
  <déclaration_variables_locales>
  begin
    corps de la méthode
  end <nom_methode>;
...
end;

```

1.4 Invocation d'une méthode

Un envoi de message ou un appel de méthode est symbolisé par un **point** :

receveur.méthode (paramètres) . La méthode est définie dans la classe du receveur.

1.5 Exemples

Exemple de définition de type	
<pre> create type Adresse as object (num Number(3), rue Varchar2(40), ville Varchar2(30), code_postal Number(5)); @compile </pre>	<pre> create type Personne as object (nom Varchar2(30), prenom Varchar2(30), habite Adresse, date_naissance Date, member function age return Number); @compile </pre>

Exemple de corps de méthode
<pre> create type body Personne as member function age return Number is begin return sysdate – date_naissance; end age; end; @compile </pre>

2. Définition du stockage

Les données sont stockées dans des tables. Définir des relations (avec les éventuelles relations imbriquées).

<Stockage> = (<définition_table>) *

<définition_table> =

```

create table <nom_table> of <nom_type>
  ( nested table <nom_attribut> store as <nom_table_imbriquée> , ) * ;

```

Inutile de préciser de *nested table* pour le type ensembliste *Varray*. En revanche, définir une table imbriquée pour chaque attribut ensembliste de type «*table of*». Toute instance possédant un identifiant d'objet doit être stockée dans une relation. Il ne peut y avoir de référence à une instance qui n'est pas stockée dans une relation. Ne pas compiler les instructions de création de table. (i.e., pas de / après un *create table*).

3. Création des données

Utiliser directement des instructions SQL *insert* ou *update* pour instancier la base.

insert into <relation> **values**(<valeur>, ...);

<valeur> = <nombre> | 'chaîne de caractère' | <objet> | variable

<objet> = <type>(<valeur>, ...) -- le constructeur <type> est le nom du type de l'objet

Utiliser une procédure PL/SQL pour traiter l'insertion : définir une procédure d'insertion, puis l'exécuter

```
create or replace procedure <nom_procédure> as
<declaration_variables_locales>
begin
...
end;
@compile
```

Compiler chaque définition de procédure individuellement avec la commande « / » ou @compile

```
begin
<nom_procédure>
end;
/
```

4. Requêtes SQL3

Forme générale d'une requête

```
select [distinct] <projection>
from <données>
[where <conditions>]
```

La clause select exprime le résultat recherché (*i.e.*, la projection). La syntaxe simplifiée est :

```
<projection> = (<attribut_proj>, ) +
<attribut_proj> = <chemin> | value(<chemin>) | ref(<chemin>) | deref(<chemin>)
<chemin> = <elt_chemin> | <elt_chemin>.<chemin>
<elt_chemin> = <variable> | <attribut> | <nom_fonction>
```

La clause from exprime l'ensemble des données accédées. La syntaxe simplifiée est :

```
<données> =
( <collections> <variable> | Table(<chemin>) <variable> )*
```

Requête avec group by

Comme en SQL on peut partitionner un ensemble suivant certains attributs de regroupement :

```
group by <attributs de regroupement>
```

Requête avec group by ... having

On peut également filtrer les groupes obtenus avec la clause **having**. La syntaxe est

```
group by <attributs de regroupement>
having <prédicat>
```

Un groupe est sélectionné ssi il satisfait le prédicat de la clause **having**. (Rmq : Ne pas confondre where avec having)

Les fonctions d'agrégat : count, sum; min, max, avg

Comme en SQL, on trouve différentes fonctions d'agrégat qui retournent une valeur numérique.

Les quantificateurs

Comme en SQL, on dispose des quantificateurs **exists** et **all**.

x.age > all 60 Cette expression rend vrai ssi le prédicat **x.age>60** est vrai **pour tout x**.

exists (sous-requête) Cette expression rend vrai ssi le résultat de la sous requête n'est pas vide.

Requêtes imbriquées

Comme en SQL, les clauses **select** , **from** et **where** peuvent contenir des sous requêtes imbriquées.

TD : XML

Modèle de données, DTD

L'objectif de ce TD est de maîtriser le langage de définition de type de données DTD, le langage de représentation des données XML. La plupart des exercices peuvent être faits sur machine à titre de préparation, avant la séance de TD.

Préparation

Lire la documentation en ligne (lien TME XML depuis la page d'accueil).

Pré-requis : connaître la syntaxe des DTD décrite dans le cours : définition d'un élément et de son contenu, définition des attributs d'un élément.

Installation des fichiers :

```
cd // aller dans le répertoire $HOME
tar zxvf $BD_TOOL/dtd.tgz // installer l'archive
cd xml // aller dans le répertoire de travail
```

Exercice 1 : TME DTD

1. DTD : validation du modèle de contenu

Soit le type de donnée T défini par :

```
<!ELEMENT XXX (AAA? , BBB+)>
<!ELEMENT AAA (CCC? , DDD*)>
<!ELEMENT BBB (CCC, DDD)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
```

Soient les 5 documents *document1.xml* à *document5.xml* suivants :

<pre><!-- document 1 --> <XXX> <AAA> <CCC/><DDD/> </AAA> <BBB> <CCC/><DDD/> </BBB> </XXX></pre>	<pre><!-- document 2 --> <XXX> <AAA> <CCC/><CCC/> <DDD/><DDD/> </AAA> <BBB> <CCC/><DDD/> </BBB> </XXX></pre>
<pre><!-- document 3 --> <XXX> <Bbb> <CCC/><DDD/> </BBB> </XXX></pre>	<pre><!-- document 5 --> <XXX> <AAA> <BBB> <CCC/><DDD/> </BBB> </XXX></pre>
<pre><!-- document 4 --> <XXX> <AAA/> <BBB/> </XXX></pre>	

- 1.1) Une DTD représente la structure en arbre d'un document XML. Quelle est la profondeur du document1 (ie., le nombre de niveaux de l'arbre). Quelle est la profondeur maximale d'un document XML respectant le type de données T ?
- 1.2) Pour chaque document, déterminer s'il est bien formé et s'il est valide pour le type de donnée T ? Expliquer chaque erreur que vous détectez.
- 1.3) Vérifiez les réponses de la question précédente en utilisant l'outil de validation.

valide documentN.xml // $N \in [1,5]$

- 1.4) Corriger les documents (en effectuant le minimum de modifications) pour qu'ils soient tous bien formés et valides.

2. Définition des éléments et des attributs d'un type de données

Un guide touristique des restaurants utilise le schéma objet-relationnel suivant :

- create type **Monument** as object (**nom** varchar2(30), **tarif** integer);
- create type **Menu** as object (**nom** varchar2(30), **prix** integer);
- create type **Menus** as varray(10) of Menu;
- create type **Ville** as object (**nom** varchar2(30), **département** varchar2(30), **plusBeauMonument** Monument);
- create type **Restaurant** as object (**nom** varchar2(30), **lieu** ref Ville, **fermeture** varchar2(30), **étoile** integer, **menu** Menus);

Pour faciliter l'échange de données entre deux serveurs de données touristiques, le format commun d'échange est défini en XML.

- 2.1) Définir une DTD (fichier *ville.dtd*) correspondant à la classe Ville du schéma objet telle que les objets complexes (ville et plus beau monument) et les attributs (nom, département et tarif) soient représentés par des éléments XML.

Ecrire le guide (*ville.xml*) conforme à la DTD *ville.dtd* , en ajoutant la ville Paris, département 75. Le plus beau monument de Paris est le Louvre dont l'entrée coûte 8 € Valider la DTD et les données du guide guide avec la commande :

valide ville.xml

--	--

- 2.2) Définir une deuxième DTD (*ville2.dtd*) correspondant au type Ville du schéma telle que :

- un élément XML représente un objet complexe (Ville)
- un attribut XML représente un attribut atomique (nom, département, tarif, ...).

Ecrire le guide (*ville2.xml*) conforme à la DTD *ville2.dtd* et contenant la ville Paris avec son plus beau monument. Valider la DTD et les données du guide avec la commande :

valide ville2.xml

--	--

- 2.3) Définir la DTD *restaurant.dtd* pour représenter un restaurant avec ses menus.

Ecrire le guide *restaurant.xml* pour qu'il contienne le restaurant universitaire de Paris, sans étoile, fermé le week-end, dont le menu unique coûte 2 €. Valider.

3. DTD à partir d'un document existant

Définir la DTD (*base1.dtd*) la plus simple possible qui valide le document *base1.xml* (**sans modifier** les données de *base1.xml*). Valider avec la commande :

valide base1.xml

4. DTD avec contrainte d'intégrité

4.1) Définir une DTD (fichier *contrainte.dtd*) qui permet de valider les contraintes suivantes :

c1 : un restaurant a toujours au moins 2 menus

c2 : une ville peut avoir un plus beau monument, certaines villes n'en ont pas.

c3 : un monument a toujours un nom et un tarif

c4 : une ville a toujours un nom et un département

c5 : un restaurant a entre 0 et 3 étoiles.

c6 : la ville d'un restaurant doit exister dans la base (*i.e.*, il doit exister un élément *ville* pour chaque nom de ville référencé dans un élément *restaurant*).

Valider votre DTD avec la commande :

valide base2.xml

4.2) Le document *base-sans-contrainte.xml* ne respectant pas les contraintes ci-dessus, vérifiez que la validation du document *base-sans-contrainte.xml* produit 6 erreurs :

valide base-sans-contrainte.xml

Pour chaque erreur, indiquez quelle est la contrainte non respectée.

4.3) (facultatif) Que faut-il modifier dans la DTD *base1.dtd* pour qu'elle soit suffisamment générale pour valider *base1.xml* et *base-sans-contrainte.xml* ?

Exercice 2 : TD DTD

1. Expressivité d'une DTD

La traduction d'un schéma objet-relationnel vers une DTD s'effectue-t-elle avec perte d'information ? Expliquer comment exprimer, dans une DTD, les concepts suivants du modèle objet-relationnel : type élémentaire, type complexe, type ensembliste, attribut, identifiant d'objet.

2. DTD générique du modèle relationnel

Question 1 : Ecrire une DTD *schema-relationnel.dtd* pour représenter un schéma relationnel quelconque. Pour cela, pensez aux définitions de schémas en SQL et adaptez sa syntaxe à la syntaxe XML.

Question 2 : Soit le schéma *Cinema* suivant :

REALISATEUR (nom, âge)

PRODUCTEUR (nom, budget)

FILM (numFilm, titre, réalisateur, producteur)

PROGRAMME (salle, jour, numFilm, nbEntrées)

titre est une clé minimale de FILM ; l'attribut *numFilm* de PROGRAMME est une clé étrangère qui référence l'attribut de même nom de FILM ; il y a deux étages 1 et 2 dans un cinéma, chacun comprenant deux salles, A et B (par défaut, la salle est 1A); l'âge et le budget ne sont pas obligatoires ; le nombre d'entrées *nbEntrees* est obligatoire.

Créer une instance de *schema-relationnel.dtd* pour décrire le schéma *Cinema*.

Question 3 : Une fois le schéma défini, il faut introduire les données dans la base. Nous voulons que les données de chaque schéma relationnel soient introduites dans un document XML qui suivra une DTD dédiée à ce schéma relationnel. Cette DTD est générée à partir de la définition du schéma (par exemple *cinema.xml*) au travers d'un ensemble de règles. Déterminez cet ensemble pour que la DTD puisse prendre en compte les contraintes d'intégrité suivantes : attribut non nul avec valeur par défaut, unicité, clé primaire composée de plusieurs attributs, intégrité référentielle, contrainte de domaine, contrainte de tuple, contrainte globale.

Créer la DTD pour le schéma *Cinéma*.

Question 4 : Finalement, écrire l'instance de *Cinema.dtd* qui permet d'introduire les tuples («A», 32), (« B », 5000) et (4, « Le fabuleux destin d'Amelie Poulain », « A », « B ») dans les relations REALISATEUR, PRODUCTEUR et FILM, respectivement.

fichier base1.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE base SYSTEM "base1.dtd">

<base>
  <restaurant nom="la tour d'argent" etoile="3" ville="Paris">
    <fermeture>dimanche et lundi</fermeture>
    <menu nom="buffet" prix="200" />
    <menu nom="gourmet" prix="300"/>
  </restaurant>

  <ville nom="Paris" departement="75">
    <plusBeauMonument nom="tour Eiffel" tarif="30"/>
  </ville>

  <restaurant nom="chez Bocuse" etoile="2" ville="Lyon">
    <fermeture>dimanche</fermeture>
    <menu nom="specialites lyonnaises" prix="200"/>
    <menu nom="vegetarien" prix="299"/>
  </restaurant>

  <restaurant nom="MacDo" etoile="0" ville="Avignon">
    <menu nom="standard" prix="35"/>
    <menu nom="enfant" prix="20"/>
    <menu nom="big" prix="49"/>
  </restaurant>

  <ville nom="Lyon" departement="69">
</ville>
```

fichier base-sans-contraainte.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE base SYSTEM "contraainte.dtd">

<base>
  <restaurant nom="la tour d'argent" etoile="4" ville="Paris">
    <fermeture>dimanche et lundi</fermeture>
    <menu nom="gourmet" prix="300"/>
  </restaurant>

  <ville nom="Paris">
    <plusBeauMonument nom="tour Eiffel" tarif="30"/>
    <plusBeauMonument nom="notre dame" tarif="50"/>
  </ville>

  <restaurant nom="chez Bocuse" etoile="2" ville="Lyon">
    <fermeture>dimanche</fermeture>
    <menu nom="spécialités lyonnaises" prix="200"/>
    <menu nom="végétarien" prix="299"/>
  </restaurant>

  <restaurant nom="MacDo" etoile="0" ville="Marseille">
    <menu nom="standard" prix="35"/>
    <menu nom="enfant" prix="20"/>
    <menu nom="big" prix="49"/>
  </restaurant>

  <ville nom="Lyon" departement="69">
</ville>

  <ville nom="Avignon" departement="84">
    <plusBeauMonument nom="le pont" />
  </ville>
</base>
```

TD/TME : XSCHEMA

L'objectif de ce TD/TME est de maîtriser le langage de définition de schéma XSchema.

Préparation

Lire la documentation en ligne (page d'accueil -> lien TME XML -> paragraphe XSchema).
Connaître la syntaxe de Xschema décrite dans le cours : définition d'un élément, d'un attribut, d'une restriction, ...

Exercice 1 : Conversion d'une DTD vers XSchema

Expliquer brièvement la méthode pour convertir une DTD en un schéma XML. Illustrer la méthode en expliquant comment obtenir le schéma *juicer.xsd* à partir de *juicer.dtd*,

juicer.dtd

```
<!ELEMENT juicers (juicer)*>
<!ELEMENT juicer (name, image, description, warranty?, weight?, cost+,
retailer)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT image (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT warranty (#PCDATA)>
<!ELEMENT weight (#PCDATA)>
<!ELEMENT cost (#PCDATA)>
<!ELEMENT retailer (#PCDATA)>
```

juicer.xml

```
<?xml version="1.0"?>

<juicers xmlns="http://www.juicers.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.juicers.org juicers.xsd">

  <juicer>
    <name>OJ Home Juicer</name>
    <image>images\mighty_oj.gif</image>
    <description>There&apos;s just no substitute for a properly
squeezed orange in the morning. So delicate and refreshing.</description>
    <warranty>lifetime warranty</warranty>
    <cost>41.95</cost>
    <retailer>http://www.thewhitewhale.com/oj.htm</retailer>
  </juicer>

  <juicer>
    <name>Wheateena Wheatgrass Juicer</name>
    <image>images\wheateena.jpg</image>
    <description>Wheatgrass juice contains 70% "crude"
chlorophyll</description>
    <warranty>Motor Guarantee: For Home Use: one (1) year. For
Commercial Use: six (6) months.</warranty>
    <weight>46</weight>
    <cost>639.99</cost>
```

```
<retailer>http://www.rawfoods.com/marketplace/heavyduty.html</retailer>
  </juicer>

</juicers>
```

juicer.xsd

```
<?xml version="1.0"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.juicers.org"
  xmlns="http://www.juicers.org"
  elementFormDefault="qualified">
  <xsd:element name="juicers">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="juicer" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="juicer">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="name"/>
        <xsd:element ref="image"/>
        <xsd:element ref="description"/>
        <xsd:element ref="warranty" minOccurs="0"/>
        <xsd:element ref="weight" minOccurs="0"/>
        <xsd:element ref="cost" maxOccurs="unbounded"/>
        <xsd:element ref="retailer"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="image" type="xsd:string"/>
  <xsd:element name="description" type="xsd:string"/>
  <xsd:element name="warranty" type="xsd:string"/>
  <xsd:element name="weight" type="xsd:string"/>
  <xsd:element name="cost" type="xsd:string"/>
  <xsd:element name="retailer" type="xsd:string"/>
</xsd:schema>
```

Exercice 2 : Modèle de contenu en DTD et XSchema

Soit un jeu de 32 cartes composé de 4 as (A), 4 rois (R), 4 dames (D), 4 valets (V), et 16 petites cartes (7, 8, 9 et 10) appelées nombre (N). Une application de jeu utilise XML pour représenter les mains de cartes. La DTD d'une main est :

```
<!ELEMENT main (A|R|D|V|N)+>
<!ELEMENT A EMPTY >
<!ELEMENT R EMPTY >
<!ELEMENT D EMPTY >
<!ELEMENT V EMPTY >
```

```
<!ELEMENT N (#PCDATA) > <!-- la feuille #PCDATA est le nombre 7,8, 9 ou 10-->
```

La règle du jeu indique qu'un joueur gagne la partie dès que sa main a 5 cartes et vaut **au moins** p points. La valeur totale d'une main est la somme des valeurs des cartes de la main. Pour cela on associe une valeur à chaque carte, quelle que soit sa couleur: l'as vaut 5 points, le roi 3 points, la dame 2 points, le valet 1 point, les autres cartes (nombres 7, 8, 9 et 10) ne valent rien.

Par exemple, les mains gagnantes possibles valant au moins 20 points sont :

4 as et 1 carte quelconque, // valeur $\geq 4*5$
 3 as et un roi, puis un roi ou une dame // valeur : $3*5 + 3 + 3$ ou $3*5 + 3 + 2$

On suppose que toute main est triée par ordre décroissant de valeur de carte (*i.e.*, dans l'ordre A, R, D, V, N).

Soit $G(p)$ l'ensemble de toutes les mains gagnantes triées, ayant 5 cartes et une valeur de main **au moins** égale à p . On a $G(p)$ contient toutes les mains de $G(p+1)$, plus toutes les mains valant p points :

- $G(23)$ contient une seule main : 4 as et un roi,
- $G(22)$ contient toutes les mains de $G(23)$, et toutes les mains valant 22 points
- ...
- $G(20)$ contient toutes les mains de $G(21)$, et toutes les mains valant 20 points. (rmq: l'exemple ci-dessus correspond à $G(20)$).
- $G(19)$ contient toutes les mains de $G(20)$, et toutes les mains valant 19 points.

Question 1 :

1.1) Existe-t-il des documents XML validant la DTD ci-dessus, mais qui ne représentent pas une main possible avec un jeu de 32 cartes. Si oui, donner deux exemples de tels documents.

1.2) Ecrire la DTD validant toutes les mains appartenant à $G(19)$, et seulement celles-ci. Toute main non gagnante (d'une valeur inférieure à 19 ou n'ayant pas 5 cartes) ne doit pas être conforme à cette DTD. Votre réponse doit être la plus factorisée possible.

Question 2 :

On propose plusieurs schémas. Pour chaque schéma répondre à la question :

Existe-t-il un p tel que le schéma valide toutes les mains de $G(p)$, et seulement celles-ci ?

Si **oui**, quelle est la valeur de p ?

Si **non** : Décrire en une seule phrase l'ensemble des mains conformes au schéma. Quelle est valeur V_{min} de la "plus faible" main conforme au schéma (*i.e.*, celle de valeur minimum) ? Quelle est la valeur V_{max} de la plus forte main conforme au schéma ?

2.1) Schéma S1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="main">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="A" minOccurs="4" maxOccurs="4"/>
        <xs:choice>
          <xs:element name="R"/>

```

```

        <xs:element name="D" />
        <xs:element name="V" />
        <xs:element name="N" />
    </xs:choice>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

2.2) Schéma S2

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="main">
    <xs:complexType>
      <xs:choice>
        <xs:sequence>
          <xs:element name="R" />
          <xs:element name="N" minOccurs="4" maxOccurs="4" />
        </xs:sequence>
        <xs:sequence>
          <xs:element name="D" />
          <xs:element name="V" />
          <xs:element name="N" minOccurs="3" maxOccurs="3" />
        </xs:sequence>
        <xs:sequence>
          <xs:element name="V" minOccurs="3" maxOccurs="3" />
          <xs:element name="N" minOccurs="2" maxOccurs="2" />
        </xs:sequence>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Question 3 Couleur et plusieurs joueurs

Pour prendre en compte la couleur des cartes, et les mains distribuées aux joueurs, on considère le schéma suivant :

```

1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3
4    <xs:element name="jeu">
5      <xs:complexType>
6        <xs:sequence maxOccurs="unbounded">
7          <xs:element ref="main" />
8        </xs:sequence>
9      </xs:complexType>
10   </xs:element>
11

```

```

12 <xs:element name="main">
13   <xs:complexType>
14     <xs:sequence>
15       <xs:element name="C" maxOccurs="unbounded">
16         <xs:complexType>
17           <xs:attribute name="nom" type="nom_carte" use="required"/>
18           <xs:attribute name="coul" type="coul_carte" use="required"/>
19         </xs:complexType>
20       </xs:element>
21     </xs:sequence>
22   </xs:complexType>
23 </xs:element>
24
25 <xs:simpleType name="nom_carte">
26   <xs:restriction base="xs:string">
27     <xs:enumeration value="A"/><xs:enumeration value="R"/>
28     <xs:enumeration value="D"/><xs:enumeration value="V"/>
29     <xs:enumeration value="10"/><xs:enumeration value="9"/>
30     <xs:enumeration value="8"/><xs:enumeration value="7"/>
31   </xs:restriction>
32 </xs:simpleType>
33 <xs:simpleType name="coul_carte">
34   <xs:restriction base="xs:string">
35     <xs:enumeration value="Pi"/><xs:enumeration value="Co"/>
36     <xs:enumeration value="Ca"/><xs:enumeration value="Tr"/>
37   </xs:restriction>
38 </xs:simpleType>
39 </xs:schema>

```

3.1) On souhaite modifier le schéma afin que les contraintes suivantes soient vérifiées :

C1: Une main a exactement 5 cartes,

C2: il y a entre 2 et 4 joueurs (avec une main par joueur).

C3: Chaque carte est unique, il ne peut y avoir deux fois la même carte de même couleur dans le jeu.

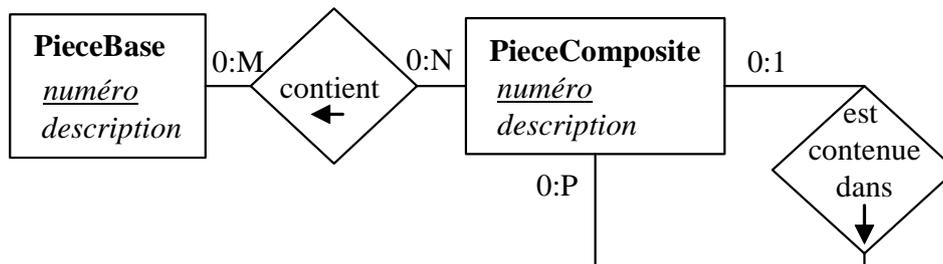
Pour chaque contrainte, indiquer le numéro des lignes à modifier, puis écrivez seulement le morceau modifié du schéma.

3.2) (question subsidiaire) On suppose maintenant que les cartes d'une main **ne sont plus triées**. Expliquer brièvement comment modifier le schéma pour qu'il valide toutes les mains de G(20) et seulement celles-ci.

3.3) (question subsidiaire) On souhaite réutiliser ce schéma pour le jeu de tarot. Expliquer brièvement comment étendre ce schéma pour représenter les cartes supplémentaires du tarot: 4 cavaliers, 21 atouts, les petites cartes de nombre (2 à 6) et le joker. Les atouts ont un numéro de 1 à 21 mais pas de couleur.

Exercice 3 : Xschéma et l'intégrité référentielle

Soit le schéma entité association suivant :



Légende: numéro est un identifiant unique

Association pièce de base – pièce composite :

- une pièce composite contient (0 à N) pièce de base,
- une pièce de base est contenue dans (0 à M) pièce composites.

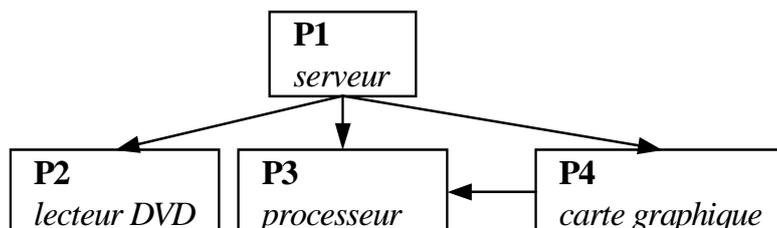
Association récursive :

- une pièce composite contient (0 à P) pièce composite,
- une pièce composite est contenue dans (0 à 1) pièce composite.

a) Transformer le diagramme E/A ci-dessus en un schéma XML tel que :

- toutes les pièces sont contenues dans l'élément racine <stock>,
- les entités sont exprimées par des éléments XML,
- le numéro est un nombre entier,
- une association 1-N est représentée par un attribut,
- une association N-M est représentée par une séquence d'éléments de type simple (*i.e.*, de type entier),
- l'intégrité référentielle est représentée par des éléments **key** et **keyref**.

b) Ecrire le document XML conforme au schéma de la question a) et contenant les pièces suivantes :



Légende : $\boxed{X} \xrightarrow{\text{contient}} \boxed{Y}$

Annexe pour la partie b

Stock.dtd

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

<!ELEMENT Stock (PC|PB)*>
<!ELEMENT PC EMPTY>
<!ATTLIST PC num ID #REQUIRED
  description CDATA #IMPLIED
  contientPC IDREFS #IMPLIED
  contientPB IDREFS #IMPLIED
  contenue_dans IDREF #IMPLIED>

<!ELEMENT PB EMPTY>
<!ATTLIST PB num ID #REQUIRED
  description CDATA #IMPLIED
  contenue_dans IDREFS #IMPLIED>
```

Exercice 4 : Schéma du guide touristique

4.1) Définir le schéma (*base.xsd*) du guide touristique (*base.xml*).

base3.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<base xmlns="Espace_TPXML"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="Espace_TPXML
    base3.xsd">

  <restaurant nom="la tour d'argent" etoile="3" ville="Paris">
    <fermeture>dimanche et lundi</fermeture>
    <menu nom="buffet" prix="200" />
    <menu nom="gourmet" prix="300"/>
  </restaurant>

  <ville nom="Paris" departement="75">
    <plusBeauMonument nom="tour Eiffel" tarif="30"/>
  </ville>

  <restaurant nom="chez Bocuse" etoile="2" ville="Lyon">
    <fermeture>dimanche</fermeture>
    <menu nom="specialites lyonnaises" prix="200"/>
    <menu nom="vegetarien" prix="299"/>
  </restaurant>

  <restaurant nom="MacDo" etoile="0" ville="Avignon">
    <menu nom="standard" prix="35"/>
```

```

    <menu nom="enfant" prix="20"/>
    <menu nom="big" prix="49"/>
  </restaurant>

  <ville nom="Lyon" departement="69">
  </ville>

  <ville nom="Avignon" departement="84">
    <plusBeauMonument nom="le pont" tarif="0"/>
  </ville>

  <restaurant nom="Les 4 saisons" etoile="1" ville="Paris">
    <fermeture>mardi</fermeture>
    <menu nom="assiette printaniere" prix="50" />
    <menu nom="salade d'ete" prix="30"/>
    <menu nom="menu d'automne" prix="150"/>
    <menu nom="menu d'hiver" prix="99"/>
  </restaurant>

  <ville nom="Brest" departement="29">
    <plusBeauMonument nom="oceanopolis" tarif="95"/>
  </ville>

</base>

```

4.2) Définir le schéma (*base-contrainte.xsd*) du guide touristique (*base-contrainte.xml*) pour vérifier les contraintes suivantes :

c1 : un restaurant a toujours au moins 2 menus

c2 : une ville peut avoir un plus beau monument, certaines villes n'en ont pas.

c3 : un monument a toujours un nom et un tarif

c4 : une ville a toujours un nom et un département

c5 : un restaurant a entre 0 et 3 étoiles.

c6 : la ville d'un restaurant doit exister dans la base (*i.e.*, il doit exister un élément *ville* pour chaque nom de ville référencé dans un élément *restaurant*). Pour définir cette contrainte, utiliser les éléments *key* et *keyref*

base-contrainte.xml

```

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

<base xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="base-contrainte.xsd">

... idem base3.xml

</base>

```

Exercice 5 : Schéma pour une librairie

5.1) Proposer un schéma (*BookStore.xsd*) pour le catalogue (*BookStore.xml*). Un élément *xs:element* doit contenir la définition de son type (attribut *type* ou sous élément *xs:complexType*).

BookStore.xml

```
<?xml version="1.0"?>

<BookStore xmlns="http://www.books.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.books.org/BookStore.xsd">

  <Book>
    <Title>My Life and Times</Title>
    <Author>Paul McCartney</Author>
    <Date>1998</Date>
    <ISBN>1-56592-235-2</ISBN>
    <Publisher>McMillin Publishing</Publisher>
  </Book>
  <Book>
    <Title>Illusions The Adventures of a Reluctant
Messiah</Title>
    <Author>Richard Bach</Author>
    <Date>1977</Date>
    <ISBN>0-440-34319-4</ISBN>
    <Publisher>Dell Publishing Co.</Publisher>
  </Book>
  <Book>
    <Title>The First and Last Freedom</Title>
    <Author>J. Krishnamurti</Author>
    <Date>1954</Date>
    <ISBN>0-06-064831-7</ISBN>
    <Publisher>Harper & Row</Publisher>
  </Book>
</BookStore>
```

5.2) Ecrire le schéma *BookStore2.xsd* en imbriquant au maximum la définition des éléments. La racine du schéma (élément *xs:schema*) ne doit posséder qu'un seul sous élément direct *xs:element*.

5.3) Ecrire le schéma *BookStore3.xsd* en séparant la définition des types et des éléments. Définir d'abord tous les types complexes (*xs:complexType*), puis définir ensuite tous les éléments.

5.4) Ecrire le schéma *BookStore4.xsd* en définissant un type *Publication*, ainsi qu'un sous-type *BookPublication* qui en hérite.

5.5) Ecrire le schéma *BookStore5.xsd* en définissant un type, *ISBN-type*, basé sur le type *String* et permettant d'imposer un format à l'élément *ISBN*.

Exercice 6 : Schéma pour juicer.xml

6.1) Ecrire le schéma juicer.xsd correspondant à la DTD juicer.dtd.

juicer.dtd

```
<!ELEMENT juicers (juicer)*>
<!ELEMENT juicer (name, image, description, warranty?, weight?, cost+,
retailer)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT image (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT warranty (#PCDATA)>
<!ELEMENT weight (#PCDATA)>
<!ELEMENT cost (#PCDATA)>
<!ELEMENT retailer (#PCDATA)>
```

juicer.xml

```
<?xml version="1.0"?>

<juicers xmlns="http://www.juicers.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.juicers.org juicers.xsd">

  <juicer>
    <name>OJ Home Juicer</name>
    <image>images\mighty_oj.gif</image>
    <description>There's just no substitute for a properly
squeezed orange in the morning. So delicate and refreshing.</description>
    <warranty>lifetime warranty</warranty>
    <cost>41.95</cost>
    <retailer>http://www.thewhitewhale.com/oj.htm</retailer>
  </juicer>

  <juicer>
    <name>Wheateena Wheatgrass Juicer</name>
    <image>images\wheateena.jpg</image>
    <description>Wheatgrass juice contains 70% "crude" chlorophyll
</description>
    <warranty>Motor Guarantee: For Home Use: one (1) year. For
Commercial Use: six (6) months.</warranty>
    <weight>46</weight>
    <cost>639.99</cost>

    <retailer>http://www.rawfoods.com/marketplace/heavyduty.html</retailer>
  </juicer>

</juicers>
```

6.2) Modifier le schéma de l'exercice 5.1 en employant XMLSchema comme namespace par défaut.

6.3) Modifier le schéma `juicers.xsd` de l'exercice 6.1 afin de le rendre plus compact en imbriquant la déclaration de tous les éléments dans l'élément `juicers`.

6.4) Modifier le schéma `juicers.xsd` afin d'utiliser des types plus significatifs que le type `String` pour la déclaration des éléments `weight`, `cost`, et `retailer`.

6.5) Modifier le schéma de l'exercice 6.4 en définissant un nouveau type, `money`, afin de l'utiliser dans la déclaration de l'élément `cost`.

XPath

TD :

Exercice 1. Restaurant

Données: le GUIDE TOURISTIQUE (voir ex précédents)

Question 1

On veut extraire des données du guide touristique *base1.xml*. Pour chaque requête, peut-on l'exprimer en une seule expression Xpath ? Si oui, donner la réponse, sinon expliquer pourquoi.

- 1) tous les menus à moins de 50 EUR
- 2) les menus des restaurants 2 ou 3 étoiles
- 3) le nom des villes dans le département 69
- 4) le nom des restaurants à Lyon
- 5) le nom des restaurants dans le département 75
- 6) le plus beau monument des villes ayant au moins 1 restaurant 3 étoiles
- 7) les villes avec au moins un restaurant qui a au moins 4 menus
- 8) les restaurants 3 étoiles fermés le dimanche
- 9) les restaurants ayant au moins un menu contenant le nom de la ville
- 10) (a) le 2ème menu de chaque restaurant
(b) le 5ème menu du guide touristique. Rmq : ce n'est pas forcément le 5ème menu du premier restaurant.
- 11) le nombre d'étoiles des restaurants qui se trouvent dans la troisième ville du documents (aide: on doit obtenir le nombre d'étoiles des restaurants à Avignon).
- 12) (a) le 2ème menu à moins de 150 EUR de chaque restaurant.
(b)Le 2ème menu de chaque restaurant s'il vaut moins de 150 EUR.
- 13) les restaurants ouverts le lundi
- 14) les villes sans restaurant 3 étoiles
- 15) (a) les villes sans plus beau monument

(b) les restaurants dans une ville sans plus beau monument

16) les noms des restaurants dont tous les menus coûtent moins cher que les menus du restaurant "Les quatre saisons".

17) les villes avec au moins deux restaurants différents classés 3 étoiles

18) les noms des restaurants proposant au moins deux menus (contenant le mot) "salade" avec des prix différents

19) La liste de tous les prix de menu possibles, chaque prix doit être listé une seule fois.

20) le nom des restaurants qui ont au moins un menu dont le prix est égal au tarif du plus beau monument de la ville (du restaurant en question) ?

21) évaluer les expressions suivantes sur le guide touristique puis décrire le résultat en une seule phrase:

- (a) /base/*/*
- (b) //*
- (c) /base/*[@* < 5]/@nom

Exercice 2. Atlas

On veut extraire des données d'un atlas conforme à la DTD *atlas.dtd*.

fichier atlas.dtd
<pre> <!-- atlas est la racine --> <!ELEMENT atlas (pays+, continent+, (montagne*, mer*)*) > <!ELEMENT pays (nom, langue+, frontière*) > <!ATTLIST pays n ID #REQUIRED continent IDREF #REQUIRED population CDATA #REQUIRED> <!-- en millions d'habitants--> <!ELEMENT nom (#PCDATA) > <!ELEMENT langue (#PCDATA) > <!ATTLIST langue pourcentage CDATA #IMPLIED> <!-- entre 0 et 100 --> <!ELEMENT frontière EMPTY > <!ATTLIST frontière pays IDREF #REQUIRED longueur CDATA #IMPLIED> <!ELEMENT continent EMPTY > <!ATTLIST continent n ID #REQUIRED nom CDATA #REQUIRED superficie CDATA #IMPLIED> <!-- en millions de km²--> <!ELEMENT montagne (situation*) > <!ATTLIST montagne n ID #REQUIRED nom CDATA #REQUIRED altitude CDATA #IMPLIED> <!-- en mètres --> <!ELEMENT mer (situation*) > <!ATTLIST mer n ID #REQUIRED nom CDATA #REQUIRED profondeur CDATA #IMPLIED> <!-- la situation représente l'appartenance d'un élément (montagne, mer, ...) à un pays --> <!ELEMENT situation EMPTY > <!ATTLIST situation pays IDREF #REQUIRED> </pre>

Pour chaque requête, peut-on l'exprimer en une expression Xpath ? Si oui, donner la réponse, sinon expliquer pourquoi et essayez de modifier la DTD afin de pouvoir trouver une expressions

Xpath pour cette question, tout en préservant au maximum la redondance (ie en évitant au maximum les répétitions de noeuds). On supposera qu'un pays n'appartient qu'à un seul continent.

Remarque : L'élément *x* désigne l'élément dont l'attribut *n* vaut *x*. Par exemple, le *pays p1* est l'élément pays dont l'attribut *n* vaut *p1*.

- 1) Les pays qui ont une frontière avec le pays *p1*
- 2) Les pays d'Europe
- 3) Les langues des pays ayant au moins 2 langues et plus de 10 millions d'habitants
- 4) Les langues que parlent 10 à 20% de la population d'un pays sur le continent *c1*.
- 5) Les pays avec au moins une montagne de sommet supérieur à 4000m
- 6) Les montagnes européennes
- 7) Les pays non maritimes (*ie.*, pour lesquels aucune mer n'appartient au pays)
- 8) Le nom des pays frontaliers de la France
- 9) Les montagnes situées sur au moins 2 pays avec chacun au moins 40 millions d'habitants
- 10) La superficie des continents possédant plus de 30 pays
- 11) (a) Les pays possédant plus de 2 montagnes
(b) Les pays possédant plus de 10 montagnes

TME:

Énoncé à consulter sur le site:

http://www-bd.lip6.fr/wiki/site/enseignement/master/mlbda/tme_xpath

TD et TME : XQuery

L'objectif de ce TD/TME est de connaître XQuery le langage d'interrogation de données XML.
Préparation : Lire la documentation en ligne (lien TME Xquery).

Exercice 1

1) Requêtes

On utilise le langage d'interrogation XQuery pour consulter un guide touristique au format XML :

Le document **guide.xml** est :

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

<base>
  <restaurant nom="la tour d'argent" etoile="3" ville="Paris">
    <fermeture>dimanche et lundi</fermeture>
    <menu nom="buffet" prix="200" />
    <menu nom="gourmet" prix="300"/>
  </restaurant>

  <ville nom="Paris" departement="75">
    <plusBeauMonument nom="tour Eiffel" tarif="30"/>
  </ville>

  <restaurant nom="chez Bocuse" etoile="2" ville="Lyon">
    <fermeture>dimanche</fermeture>
    <menu nom="specialites lyonnaises" prix="200"/>
    <menu nom="vegetarien" prix="299"/>
  </restaurant>

  <restaurant nom="MacDo" etoile="0" ville="Avignon">
    <menu nom="standard" prix="35"/>
    <menu nom="enfant" prix="20"/>
    <menu nom="big" prix="49"/>
  </restaurant>

  <ville nom="Lyon" departement="69">
  </ville>

  <ville nom="Avignon" departement="84">
    <plusBeauMonument nom="le pont" tarif="0"/>
  </ville>

  <restaurant nom="Les 4 saisons" etoile="1" ville="Paris">
    <fermeture>mardi</fermeture>
    <menu nom="assiette printaniere" prix="50" />
    <menu nom="salade d'ete" prix="30"/>
    <menu nom="menu d'automne" prix="150"/>
    <menu nom="menu d'hiver" prix="99"/>
  </restaurant>

  <ville nom="Brest" departement="29">
    <plusBeauMonument nom="oceanopolis" tarif="95"/>
  </ville>
</base>
```

Définir en XQuery les requêtes suivantes (une seule requête par fichier).

1.1) req1.xql : Donner le nom de tous les menus des restaurants

Résultat:

```
<results>
  <menu nom="buffet"/>
```

```

<menu nom="gourmet"/>
<menu nom="specialites lyonnaises"/>
<menu nom="vegetarien"/>
<menu nom="standard"/>
<menu nom="enfant"/>
<menu nom="big"/>
<menu nom="assiette printaniere"/>
<menu nom="salade d'ete"/>
<menu nom="menu d'automne"/>
<menu nom="menu d'hiver"/>
</results>

```

1.2) req2.xql : Donner le nom et le prix de tous les menus dont le prix est inférieur à 100.

Résultat:

```

<results>
  <menu nom="standard" prix="35"/>
  <menu nom="enfant" prix="20"/>
  <menu nom="big" prix="49"/>
  <menu nom="assiette printaniere" prix="50"/>
  <menu nom="salade d'ete" prix="30"/>
  <menu nom="menu d'hiver" prix="99"/>
</results>

```

1.3) req3.xql : Donner le nom des restaurants 2 étoiles avec leur nom de menu

Résultat:

```

<results>
  <restaurant nom="chez Bocuse">
    <menu nom="specialites lyonnaises"/>
    <menu nom="vegetarien"/>
  </restaurant>
</results>

```

1.4) req4.xql : Donner le nom de chaque restaurant avec son numéro de département

Résultat:

```

<results>
  <restaurant nom="la tour d'argent" departement="75"/>
  <restaurant nom="chez Bocuse" departement="69"/>
  <restaurant nom="MacDo" departement="84"/>
  <restaurant nom="Les 4 saisons" departement="75"/>
</results>

```

1.5) req5.xql : Quels sont les restaurants ayant (au moins) un menu dont le prix est égal au tarif de visite du plus beau monument de la ville ? Donner le nom du restaurant et le tarif.

Résultat:

```

<results>
  <result>
    <restaurant nom="Les 4 saisons"/>
    <tarif_monument prix="30"/>
  </result>
</results>

```

Pour chaque requête, donner la DTD pour valider le résultat (fichier reqN.dtd).

Exercice 2

Les données manipulées dans cet exercice sont représentées en XML. Il y a 4 types de données : les références bibliographiques (bib), les critiques (review) et les prix comparatifs (price) et les livres (book). Les données sont définies comme suit :

Les références bibliographiques

Le document **bib.xml** est :

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price> 65.95</price>
  </book>
  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="2000">
    <title>Data on the Web</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Buneman</last><first>Peter</first></author>
    <author><last>Suciu</last><first>Dan</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price> 39.95</price>
  </book>
  <book year="1999">
    <title>The Economics of Technology and Content for Digital TV</title>
    <editor>
      <last>Gerbarg</last><first>Darcy</first>
      <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>
```

Les critiques de livres

Le document **review.xml** est :

```
<reviews>
  <entry>
    <title>Data on the Web</title>
    <price>34.95</price>
    <review>
      A very good discussion of semi-structured database
      systems and XML.
    </review>
  </entry>
  <entry>
    <title>Advanced Programming in the Unix environment</title>
    <price>65.95</price>
    <review>
      A clear and detailed discussion of UNIX programming.
    </review>
  </entry>
  <entry>
    <title>TCP/IP Illustrated</title>
    <price>65.95</price>
    <review>
      One of the best books on TCP/IP.
    </review>
  </entry>
</reviews>
```

Les prix comparatifs :

La DTD **price.dtd** est :

Le document **price.xml** est :

<pre> <!ELEMENT prices (book*)> <!ELEMENT book (title, source, price)> <!ELEMENT title (#PCDATA)> <!ELEMENT source (#PCDATA)> <!ELEMENT price (#PCDATA)> </pre>	<pre> <prices> <book> <title>Advanced Programming in the Unix environment</title> <source>www.amazon.com</source> <price>65.95</price> </book> <book> <title>Advanced Programming in the Unix environment </title> <source>www.bn.com</source> <price>65.95</price> </book> <book> <title>TCP/IP Illustrated</title> <source>www.amazon.com</source> <price>65.95</price> </book> <book> <title>TCP/IP Illustrated</title> <source>www.bn.com</source> <price>65.95</price> </book> <book> <title>Data on the Web</title> <source>www.amazon.com</source> <price>34.95</price> </book> <book> <title>Data on the Web</title> <source>www.bn.com</source> <price>39.95</price> </book> </prices> </pre>
---	--

Les livres

<p>La DTD book.dtd est :</p> <pre> <!ELEMENT chapter (title, section*)> <!ELEMENT section (title, section*)> <!ELEMENT title (#PCDATA)> </pre>	<p>Le document book.xml est :</p> <pre> <chapter> <title>Data Model</title> <section> <title>Syntax For Data Model</title> </section> <section> <title>XML</title> <section> <title>Basic Syntax</title> </section> <section> <title>XML and Semistructured Data</title> </section> </section> </chapter> </pre>
---	---

Ecrire les requêtes suivantes en XQuery

R1 : Sélection

Donner l'année et le titre des livres publiés par Addison Wesley après 1991.

Résultat:

<pre> <bib> <book year="1994"> <title>TCP/IP Illustrated</title> </pre>

```

</book>
<book year="1992">
  <title>Advanced Programming in the Unix environment</title>
</book>
</bib>

```

R2 : Structure plate et désimbriquée

Donner la liste de tous les couples (titre, auteur). Un couple est formé par un élément <result> contenant un titre et un auteur.

Résultat:

```

<results>
  <result>
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
  </result>
  <result>
    <title>Advanced Programming in the Unix environment</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
  </result>
  <result>
    <title>Data on the Web</title>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first>
    </author>
  </result>
  <result>
    <title>Data on the Web</title>
    <author>
      <last>Buneman</last>
      <first>Peter</first>
    </author>
  </result>
  <result>
    <title>Data on the Web</title>
    <author>
      <last>Suciu</last>
      <first>Dan</first>
    </author>
  </result>
</results>

```

R3 : Structure hiérarchique

Pour chaque livre, donner le titre et les auteurs regroupés dans un élément <result>.

Résultat:

```

<results>
  <result>
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>

```

```

</author>
</result>
<result>
  <title>Advanced Programming in the Unix environment</title>
  <author>
    <last>Stevens</last>
    <first>W.</first>
  </author>
</result>
<result>
  <title>Data on the Web</title>
  <author>
    <last>Abiteboul</last>
    <first>Serge</first>
  </author>
  <author>
    <last>Buneman</last>
    <first>Peter</first>
  </author>
  <author>
    <last>Suciu</last>
    <first>Dan</first>
  </author>
</result>
<result>
  <title>The Economics of Technology and Content for Digital TV</title>
</result>
</results>

```

R4 : Auto jointure d'un document

Pour chaque auteur, donner la liste des titres de ses livres. Un élément du résultat contient un auteur avec tous les titres qu'il a écrit.

Résultat:

```

<results>
  <result>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <title>TCP/IP Illustrated</title>
    <title>Advanced Programming in the Unix environment</title>
  </result>
  <result>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first>
    </author>
    <title>Data on the Web</title>
  </result>
  <result>
    <author>
      <last>Buneman</last>
      <first>Peter</first>
    </author>
    <title>Data on the Web</title>
  </result>
  <result>
    <author>
      <last>Suciu</last>
      <first>Dan</first>
    </author>
    <title>Data on the Web</title>
  </result>

```

</results>

R5 : Jointure entre deux documents

Pour chaque livre de la bibliographie Bib qui a une critique dans Review, donner son titre et tous ses prix.

Résultat:

```
<books-with-prices>
  <book-with-prices>
    <title>TCP/IP Illustrated</title>
    <price-review>65.95</price-review>
    <price-bib>65.95</price-bib>
  </book-with-prices>
  <book-with-prices>
    <title>Advanced Programming in the Unix environment</title>
    <price-review>65.95</price-review>
    <price-bib>65.95</price-bib>
  </book-with-prices>
  <book-with-prices>
    <title>Data on the Web</title>
    <price-review>34.95</price-review>
    <price-bib>39.95</price-bib>
  </book-with-prices>
</books-with-prices>
```

R6 : Expression conditionnelle

Pour chaque livre ayant au moins un auteur, donner le titre et le nom d'au plus deux auteurs. Le nom des auteurs suivants est remplacé par l'élément <et-al> (*i.e.*, abréviation latine signifiant *et les autres*).

Résultat:

```
<bib>
  <book>
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
  </book>
  <book>
    <title>Advanced Programming in the Unix environment</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
  </book>
  <book>
    <title>Data on the Web</title>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first>
    </author>
    <author>
      <last>Buneman</last>
      <first>Peter</first>
    </author>
    <et-al/>
  </book>
</bib>
```

R7 : Tri

Donner, dans l'ordre alphabétique, le titre et l'année des livres publiés par Addison-Wesley après 1991.

Résultat:

```
<bib>
  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
  </book>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
  </book>
</bib>
```

R8 : Comparaison de chaînes de caractères

Quels sont les livres qui ont un élément dont le nom se termine par «or» et dont un sous élément contient la chaîne «Suciu».

Résultat:

```
<book>
  <title>Data on the Web</title>
  <author>
    <last>Suciu</last>
    <first>Dan</first>
  </author>
</book>
```

R9 : Recherche plein texte

Dans le livre *book.xml*, quels sont tous les titres de chapitre ou de section qui contiennent le mot « XML » ?

La DTD du livre *book.xml* est

```
<!ELEMENT chapter (title, section*)>
  <!ELEMENT section (title, section*)>
  <!ELEMENT title (#PCDATA)>
```

R10 :

Dans le document *price.xml*, donner le prix le moins cher de chaque livre. Le résultat est une liste d'éléments <minprice>. Le titre est un attribut de <minprice>, le prix est un élément de <minprice>.

Résultat:

```
<results>
  <minprice title="Advanced Programming in the Unix environment">
    <price>65.95</price>
  </minprice>
  <minprice title="TCP/IP Illustrated">
    <price>65.95</price>
  </minprice>
  <minprice title="Data on the Web">
    <price>34.95</price>
  </minprice>
</results>
```

R11 :

Pour chaque livre avec au moins un auteur, donner le titre et les auteurs, pour chaque livre avec un éditeur, donner une référence contenant le titre du livre et l'affiliation de l'éditeur.

Résultat:

```
<bib>
  <book>
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
  </book>
```

```

<book>
  <title>Advanced Programming in the Unix environment</title>
  <author>
    <last>Stevens</last>
    <first>W.</first>
  </author>
</book>
<book>
  <title>Data on the Web</title>
  <author>
    <last>Abiteboul</last>
    <first>Serge</first>
  </author>
  <author>
    <last>Buneman</last>
    <first>Peter</first>
  </author>
  <author>
    <last>Suciu</last>
    <first>Dan</first>
  </author>
</book>
<reference>
  <title>The Economics of Technology and Content for Digital TV</title>
  <affiliation>CITI</affiliation>
</reference>
</bib>

```

R12 :

Trouver les paires de livres qui ont les mêmes auteurs et des titres différents. Ne pas tenir compte de l'ordre des auteurs.

Résultat:

```

<bib>
  <book-pair>
    <title>TCP/IP Illustrated</title>
    <title>Advanced Programming in the Unix environment</title>
  </book-pair>
</bib>

```

Exercice 3. Extrait de l'examen de Juin2003

Les données manipulées dans cet exercice sont représentées en XML. Il y a deux types de données : des joueurs de tennis et des résultats de rencontres. Les données sont définies comme suit :

Les joueurs de tennis

Le document `joueur.xml` est :

```
<joueurs>
<joueur>
  <identite>
    <nom>Hewitt</nom>
    <prenom>Lleyton</prenom>
    <nationalite>AUS</nationalite>
  </identite>
  <classement>1</classement>
</joueur>
<joueur>
  <identite>
    <nom>Arthurs</nom>
    <prenom>Wayne</prenom>
    <nationalite>AUS</nationalite>
  </identite>
  <classement>55</classement>
</joueur>
<joueur>
  <identite>
    <nom>Mathieu</nom>
    <prenom>Paul-Henry</prenom>
    <nationalite>FR</nationalite>
  </identite>
  <classement>44</classement>
</joueur>
<joueur>
  <identite>
    <nom>Clement</nom>
    <prenom>Arnaud</prenom>
    <nationalite>FR</nationalite>
  </identite>
  <classement>37</classement>
</joueur>
<joueur>
  <identite>
    <nom>Costa</nom>
    <prenom>Albert</prenom>
    <nationalite>ESP</nationalite>
  </identite>
  <classement>8</classement>
</joueur>
<joueur>
  <identite>
    <nom>Gasquet</nom>
    <prenom>Richard</prenom>
    <nationalite>FR</nationalite>
  </identite>
  <classement>124</classement>
</joueur>
</joueurs>
```

Les rencontres

Le document **rencontre.xml** est :

```

<rencontres>
<rencontre>
  <joueur1>
    <identite>
      <nom>Hewitt</nom>
      <prenom>Lleyton</prenom>
      <nationalite>AUS</nationalite>
    </identite>
  </joueur1>
  <joueur2>
    <identite>
      <nom>Arthurs</nom>
      <prenom>Wayne</prenom>
      <nationalite>AUS</nationalite>
    </identite>
  </joueur2>
  <gagnant>1</gagnant>
</rencontre>

<rencontre>
  <joueur1>
    <identite>
      <nom>Arthurs</nom>
      <prenom>Wayne</prenom>
      <nationalite>AUS</nationalite>
    </identite>
  </joueur1>
  <joueur2>
    <identite>
      <nom>Hewitt</nom>
      <prenom>Lleyton</prenom>
      <nationalite>AUS</nationalite>
    </identite>
  </joueur2>
  <gagnant>1</gagnant>
</rencontre>

<rencontre>
  <joueur1>
    <identite>
      <nom>Hewitt</nom>
      <prenom>Lleyton</prenom>
      <nationalite>AUS</nationalite>
    </identite>
  </joueur1>
  <joueur2>
    <identite>
      <nom>Mathieu</nom>
      <prenom>Paul-Henry</prenom>
      <nationalite>FR</nationalite>
    </identite>
  </joueur2>
  <gagnant>2</gagnant>
</rencontre>

<rencontre>
  <joueur1>
    <identite>
      <nom>Clement</nom>
      <prenom>Arnaud</prenom>
      <nationalite>FR</nationalite>
    </identite>
  </joueur1>
  <joueur2>
    <identite>
      <nom>Mathieu</nom>
      <prenom>Paul-Henry</prenom>
      <nationalite>FR</nationalite>
    </identite>
  </joueur2>

```

```
<gagnant>2</gagnant>
</rencontre>
</rencontres>
```

Ecrire les requêtes suivantes en XQuery :

R1 : Structure aplatie et désimbriquée

Lister les joueurs avec leur classement et ordonnés par nationalité.

Résultat :

```
<resultats>
  <resultat>
    <nom>Hewitt</nom>
    <prenom>Lleyton</prenom>
    <classement>1</classement>
    <nationalite>AUS</nationalite>
  </resultat>
  <resultat>
    <nom>Arthurs</nom>
    <prenom>Wayne</prenom>
    <classement>55</classement>
    <nationalite>AUS</nationalite>
  </resultat>
  <resultat>
    <nom>Costa</nom>
    <prenom>Albert</prenom>
    <classement>8</classement>
    <nationalite>ESP</nationalite>
  </resultat>
  <resultat>
    <nom>Mathieu</nom>
    <prenom>Paul-Henry</prenom>
    <classement>44</classement>
    <nationalite>FR</nationalite>
  </resultat>
  <resultat>
    <nom>Clement</nom>
    <prenom>Arnaud</prenom>
    <classement>37</classement>
    <nationalite>FR</nationalite>
  </resultat>
  <resultat>
    <nom>Gasquet</nom>
    <prenom>Richard</prenom>
    <classement>124</classement>
    <nationalite>FR</nationalite>
  </resultat>
</resultats>
```

R2 : Rebalisage

Lister les joueurs en donnant leur nom, prénom et nationalité séparés par des virgules.

Résultat :

```
<resultats>
  <resultat>Hewitt,Lleyton,AUS</resultat>
  <resultat>Arthurs,Wayne,AUS</resultat>
```

```
<resultat>Mathieu, Paul-Henry, FR</resultat>
<resultat>Clement, Arnaud, FR</resultat>
<resultat>Costa, Albert, ESP</resultat>
<resultat>Gasquet, Richard, FR</resultat>
</resultats>
```

R3 : Agrégation

Donner le nombre de joueurs par nationalité.

Résultat :

```
<resultats>
  <resultat><nationalite>AUS</nationalite><nb_joueurs>2</nb_joueurs></resultat>
  <resultat><nationalite>FR</nationalite><nb_joueurs>3</nb_joueurs></resultat>
  <resultat><nationalite>ESP</nationalite><nb_joueurs>1</nb_joueurs></resultat>
</resultats>
```

R4 : Jointure

Pour chacune des rencontres, donner le classement des deux joueurs. Seules les rencontres entre joueurs de classement inférieur à 50 sont conservées.

Résultat :

```
<resultats>
  <resultat><classement>1</classement><classement>44</classement></resultat>
  <resultat><classement>37</classement><classement>44</classement></resultat>
</resultats>
```

TD et TME : RDF et SPARQL

L'objectif de ce TD/TME est de se familiariser avec la syntaxe de représentation des données RDF Turtle et de comprendre le mécanisme des motifs de graphes qui sont à la base du langage d'interrogation Sparql.

Exercice 1 : Dépliage des données factorisées

Représenter les données suivantes sous format Turtle en des triplets n'utilisant pas de factorisation.

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

<#green-goblin>
  rel:enemyOf <#spiderman> ;
  a foaf:Person ; # in the context of the Marvel universe
  foaf:name "Green Goblin" .
<#spiderman>
  rel:enemyOf <#green-goblin> ;
  a foaf:Person ;
  foaf:name "Spiderman", "Человек-паук"@ru .
```

Exercice 2 : Motifs de graphes

Le but de cet exercice est d'illustrer les principaux motifs de graphes utilisés dans le langage SPARQL. Rappelons qu'une requête SPARQL est générée à partir de la syntaxe suivante

```
SELECT [distinct] | CONSTRUCT | ASK | DESCRIBE
FROM source
WHERE { MOTIFS }
ORDER | LIMIT | OFFSET
```

Pour cet exercice, nous nous focalisons sur la partie **MOTIFS** des requêtes et considérons que leur évaluation retourne toutes variables qui y sont exprimées.

Par exemple : l'évaluation du motif `?X rel:enemyOf ?y` sur les triplets de l'exercice précédent retourne les liaisons indiquées dans le tableau suivant

?X	?Y
#green-goblin	#spiderman
#spiderman	#green-goblin

Les données de l'encadré ci-dessous sont celles qu'on utilisera dans cet exercice. Pour simplifier, les espaces de noms ainsi que le symbole ':' en préfixe des IRI sont omis.

P1	name	"john"
P2	name	"rick"
P3	name	"lars"
P4	name	"liz"
P1	phone	11-345-89
P2	email	rick@org
P3	webpage	www.lars.com
P4	email	liz@aut.gov
P4	phone	09-134-21
P4	webpage	www.liz.pers

Remarques :

- 1- Les motifs doivent toujours être séparés par des points. (cf point a ci-dessous)
- 2- Dans Sparql, un groupe de motifs {M1. M2. M3} signifie la conjonction de ces motifs. Ceci explique l'absence d'un opérateur spécifique pour exprimer la conjonction.
- 3- L'opérateur UNION prend comme opérands deux groupes de motifs qui doivent **obligatoirement** être entourés d'accolades { } même si la cardinalité des groupes est 1 (cf point b ci-dessous)
- 4- l'UNION est plus prioritaire par rapport à la conjonction des groupes de motifs.

Evaluer sur ces triplets les motifs suivants et expliquer en langage naturel ce qu'il permettent de calculer. On supposera une sémantique ensembliste.

Remarque : Pour tester ces requêtes en TME, il suffit de rajouter la clause select avec le symbole * et d'imbriquer les motifs dans la clause where.

1.
 - a. ?p email ?e . ?p webpage ?w
 - b. {?p email ?e} UNION {?p webpage ?w}
 - c. {?p email ?e} OPTIONAL {?p webpage ?w}
2. ?p name ?n OPTIONAL {?p email ?e} OPTIONAL {?p webpage ?w}
3. ?p name ?n OPTIONAL { ?p email ?e OPTIONAL {?p webpage ?w} }

Y-a-t-il une différence entre les motif des questions 2 et 3 ? Si oui laquelle?

Discussion : Que remarquez-vous en comparant les réponses des motifs des questions 2 et 3? Est-ce qu'en général on peut dire que l'opérateur OPTIONAL est associatif, i.e

{M1 OPTIONAL M2} OPTIONAL M3 = M1 OPTIONAL { M2 OPTIONAL M3} où M1, M2 et M3 sont des motifs?

En cas de réponse négative, quelle modification mineure (ajout ou suppression de quelques triplets) permettrait d'avoir pour les données de l'exercice un résultat identique en évaluant les motifs des questions 2 et 3?

4. ?p name ?n . { {?p email ?e} UNION {?p phone ?l} }
5.
 - a. ?p webpage ?w OPTIONAL {?p phone ?l} FILTER (! bound(?l))
 - b. ?p webpage ?w minus {?p phone ?l}

Remarque : Bien que Sparql n'oblige pas que les variables utilisées par la fonction bound() aient été utilisées dans le groupe de motif qui précède le filter, il est plus sûr de respecter cette contrainte.

Exemple :

le motif

```
?p webpage ?w OPTIONAL {?p phone ?l} FILTER ( ! bound(?k))
```

retourne les liaisons décrites comme suit

?p	?w	?l
P4	"www.liz.pers"	"09-134-21"
P3	"www.lars.com"	

Exercice 3 : Requêtes

Considérer les données suivantes qui portent sur des personnes.

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://www.perceive.net/schemas/relationship/> .
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix univ: <http://www.faculty.ac> .
```

#rdf types

```
:john a :Person .
:liz a :Person .
:robert a :Person .
:dan a :Person .
:J.Horrow a :Professor .
:larry a :Artist .
:cmu a :university .
:mit a :institute .
:mi a :state .
:nyc a :City .
:pittsburgh a :City .
```

#family and friendships

```
:john :hasMother :suzan .
:john :hasFather :richard .
:richard :hasBrother :luke .
:john :hasBrother :richard .
:larry :hasFather :john .
:liz :hasBrother :robert .
:liz :hasFather :luke .
:liz :hasMother :monica .
:larry :friend :liz .
:dan rel:ChildOf univ:J.Horrow .
:Jim rel:ChildOf univ:L.Yang .
:J.Horrow :studiedAt :ucla .

#POI
:robert :livesIn :nyc .
:liz :livesIn :westLafayette .

#degree and university
:luke :hasDegree "Eng" .
:luke :studiedAt :cmu .
:liz :studiedAt :cmu .
:richard :studiedAt :mit .
:suzan :studiedAt :ucsd .
:monica :supervisedBy univ:L.Yang .
:monica :supervisedBy univ:J.Horrow .
:Jim :studiedAt :ucsb .
:monica :studiedAt :cmu .
:john :studiedAt :ucsd .
:larry :studiedAt :ucsd .

#misc descriptions
:ucsb :locatedAt :santaBarbara .
:cmu :locatedAt :pittsburgh .
```

Quel est le format utilisé? Représentez les trois premiers triplets sous leur forme *étendue*.
Exprimer les requêtes suivantes :

1.
 - a. Extraire l'ensemble des IRI des sujets
 - b. idem en retournant les noms locaux

Forme du résultat attendu :

a)	b)
<pre> x ===== <http://example.org/john> <http://example.org/mi> <http://example.org/cmu> <http://example.org/richard> </pre>	<pre>----- x ===== :john :dan :mi :cmu </pre>

2. Même question avec les prédicats.
3. Extraire les villes citées dans des triples de cette base.
4. Extraire les personnes qui ont étudié dans la même université que l'un de leur parents. (Ne pas considérer les triplets avec la prédicat childOf).
5. Extraire les personnes qui ont étudié dans une université où leur deux parents ont étudié.
6. Extraire les personnes qui ont étudié dans une université où aucun de leur parent n'a étudié.
7. Extraire les personnes qui ont un étudié dans la même université qu'un frère ou une soeur.

(Question subsidiaire) Donnez les noms et les universités des personnes qui ont au moins un frère ou une soeur et qui n'ont pas de frère ou de soeur qui ont étudié à la même université qu'elles.

8. Extraire les personnes qui étudient dans une ville différente de celle où ils habitent.
9. Extraire les personnes qui sont ami(e)s d'un(e) ami(e) de Liz.
Remarque : ne pas retourner les ami(e)s direct(e)s de Liz!

Exercice 4 : Requêtes

Considérons les données RDF dans l'encadré.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

```

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix f: <http://www.cems.uwe.ac.uk/empdept/concept/> .
@prefix : <http://www.abc.org/> .

#rdf types
:john    a      f:emp .
:liz     a      f:emp .
:dan     a      f:emp .
:larry   a      f:emp .
:jim     a      f:emp .
:clark   a      f:emp .
:robert  a      f:emp .

#job
:john f:Job "architect" .
:liz  f:Job "doctor" .
:dan  f:Job "engineer" .
:larry f:Job "singer" .

#surnames
:john  foaf:surname "john" .
:liz   foaf:surname "liz" .
:dan   foaf:surname "dan" .
:larry foaf:surname "larry" .

```

Exprimer les requêtes retournant les informations suivantes

1. Noms des employés avec leur job suivant l'ordre alphabétique de leur nom
 2. Les 3 premiers employés suivant l'ordre alphabétique
 3. Les 3 premiers employés, ayant un surname, les mieux payés
 4. Les employés qui ne sont ni médecin ni chanteur
 5. Les employés dont le nom commence par 'l' et termine par 'ry'.
-
6. Le plus grand salaire
 - a. sans utiliser ORDER BY et LIMIT
 - b. en utilisant ORDER BY et LIMIT
 7. Les couples d'employés qui gagnent le même salaire (chaque couple d'employés doit apparaître une seule fois).
 8. Les couples d'employés tels que la différence entre leurs salaires est supérieure à 5.
 9. Le nombre de départements
 10. Le nombre d'employés par département