Examen MIF04 - Gestion de données pour le Web - session 1 - 3 janvier 2017
Durée : 1h30
Documents autorisés

Numéro de copie :

Il faut rendre ces feuilles en les glissant dans votre copie anonyme. Ne pas l'utiliser comme brouillon. Les réponses sont à donner sur ces feuilles, pas dans la copie. Remplir le champ ci-dessus avec le *numéro de la copie* dans laquelle vous allez la glisser. Remplir la partie d'anonymat de la copie, puis coller le coin. Le barème est donné à titre indicatif, l'examen sera noté sur 20.

Exercice 1: Correspondance entre modèles relationel, XML et objet (7 points)

On considère le modèle relationel suivant :

```
1  CREATE TABLE client(
2     id INTEGER PRIMARY KEY,
3     nom VARCHAR(255)
4  );
5  CREATE TABLE chambre(
6     numero INTEGER PRIMARY KEY,
7     prix DOUBLE
8  );
9  CREATE TABLE reservation(
10    debut DATE,
11    fin DATE,
12    client_id INTEGER REFERENCES client(id),
13    chambre_numero INTEGER REFERENCES chambre(numero)
14 );
```

On considère la DTD suivante :

```
1  <!DOCTYPE reservations [
2  <!ELEMENT reservations (client+,chambre+,reservation+)>
3  <!ELEMENT client (#PCDATA)>
4  <!ATTLIST client id ID #REQUIRED>
5  <!ELEMENT chambre (#PCDATA)>
6  <!ATTLIST chambre numero ID #REQUIRED>
7  <!ELEMENT reservation (debut,fin)>
8  <!ATTLIST reservation client IDREF #REQUIRED
9                        chambre IDREF #REQUIRED>
10 <!ELEMENT debut (#PCDATA)>
11 <!ELEMENT fin (#PCDATA)>
12 ]>
```

On considère le code Java suivant :
Classe `Client`

```java
package exercice1;

import javax.persistence.*;
import java.util.*;

@Entity
@Table(name = "client")
public class Client {

    @OneToMany
    @ManyToOne
    @ManyToMany
    @Id
    public int id

    @OneToMany
    @ManyToOne
    @ManyToMany
    @Id
    public String nom;

    @OneToMany
    @ManyToOne
    @ManyToMany
    @Id
    public Collection<Reservation> reservations;
}
```

Classe Chambre

```java
package exercice1;

import javax.persistence.*;
import java.util.*;

@Entity
@Table(name = "chambre")
public class Chambre {
    @OneToMany
    @ManyToOne
    @ManyToMany
    @Id
    public int numero;

    @OneToMany
    @ManyToOne
    @ManyToMany
    @Id
    public double prix;
```

```
20
21      @OneToMany
22      @ManyToOne
23      @ManyToMany
24      public Collection<Reservation> reservations;
25  }
```

Classe Reservation

```
1   package exercice1;
2
3   import javax.persistence.*;
4   import java.util.*;
5
6   @Entity
7   @Table(name = "reservation")
8   public class Reservation {
9       @OneToMany
10      @ManyToOne
11      @ManyToMany
12      @Id
13      @Temporal(TemporalType.DATE)
14      public Date debut;
15
16      @OneToMany
17      @ManyToOne
18      @ManyToMany
19      @Id
20      @Temporal(TemporalType.DATE)
21      public Date fin;
22
23      @OneToMany
24      @ManyToOne
25      @ManyToMany
26      @Id
27      public Client client;
28
29      @OneToMany
30      @ManyToOne
31      @ManyToMany
32      @Id
33      public Chambre chambre;
34  }
```

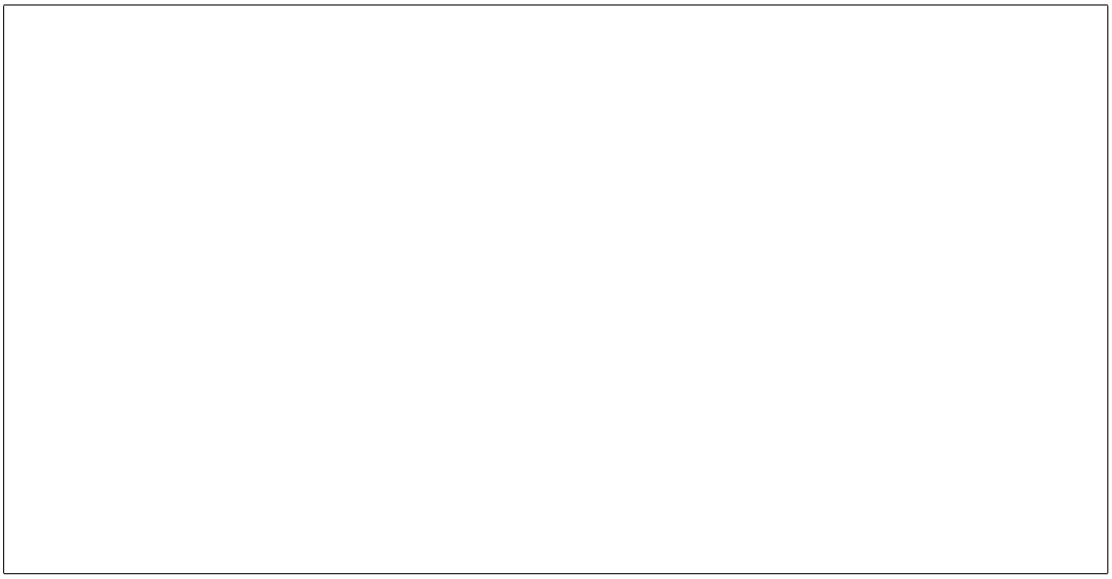
1. Rayer dans le code Java précédent les *annotations* qui vont nécessairement provoquer une erreur (sachant que les annotations `@Temporal` ne provoqueront pas d'erreur).
2. Que manque-t-il pour que la classe `Reservation` puisse être mise en correspondance ?

   Proposer une correction pour que la correspondance puisse se mettre en place.
3. On considère que les réservations sont exportées dans un document XML valide vis-à-vis de la DTD. Traduire la requête XPath suivante en une requête SQL :

```
1 /reservations/client[@id =
2     /reservations/reservation[@chambre =
3         /reservations/chambre[prix < 100]/@numero]/@client ]
```

4. Traduire en une requête XPath (à exécuter sur l'export XML) la requête SQL suivante :

```sql
SELECT prix
FROM chambre, reservation, client
WHERE numero = chambre_numero
  AND id = client_id
  AND nom = "Toto"
  AND debut <= "2017-01-03"
  AND fin >= "2017-01-03"
```



Exercice 2: RDF et RDFS (6 points)

On considère le graphe RDF suivant (format Turtle) :

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix hotel: <http://example.com/hotels/> .

hotel:dch1 hotel:equipe hotel:ch1;
           rdf:type hotel:douche.
hotel:tv1 hotel:equipe hotel:ch1;
          rdf:type hotel:television.
hotel:clim1 hotel:equipe hotel:ch2.
hotel:ch1 hotel:jouxte hotel:ch2.
hotel:equipe rdfs:domain hotel:equipement.
hotel:jouxte rdfs:domain hotel:chambre.
```

On rappelle la règle RDFS suivante :

$$(RDFSDomain) \quad \frac{P \texttt{ rdfs:domain } T \qquad S\ P\ O}{S \texttt{ rdf:type } T}$$

On donne également la règle métier suivante :

$$(JouxteS) \quad \frac{C \texttt{ hotel:jouxte } D}{D \texttt{ hotel:jouxte } C}$$

On s'intéresse à la requête SPARQL suivante :

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3 PREFIX hotel: <http://example.com/hotels/> .
4
5 SELECT ?c, ?e WHERE {
6   ?e hotel:equipe ?c2.
7   ?c2 hotel:jouxte ?c.
8   ?e rdf:type hotel:equipement.
9 }
```

1. Donner la signification de la règle $(JouxteS)$ vis-à-vis de la relation `hotel:jouxte`.

2. Saturer le graphe à l'aide de ces deux règles. Donner la liste des triplets déduits.

3. Donner la ou les lignes qui peuvent être supprimées de la requête sans en changer le résultat sur le graphe donné.

4. Evaluer la requête sur le graphe saturé.

?c	?e

5. Réécrire la requête de façon à obtenir le même résultat mais à partir du graphe non saturé.

Exercice 3: MongoDB (7 points)

On considère une collection `factures` dont un des enregistrements est donné ci-dessous :

```
{
  "id" : "332",
  "nuitees": [
     { "chambre" : 32,
       "nuits": 5,
       "montant_ht": 300.0 },
     { "chambre" : 34,
       "nuits": 5,
       "montant_ht: 250.0 }
  ],
  "total_ttc" : 664.0,
  "date" : ISODate("2017-01-03T00:00:00Z"),
  "taxe_sejour" : 4.0
}
```

Remarque : les dates peuvent être créées dans MongoDB en Javascript via `new Date("<YYYY-mm-dd>")`. Par exemple, la date du document précédent peut être générée via `new Date("2017-01-03")`.

On donne l'exemple suivant de fonction `map` et `reduce` en MongoDB :

```
map_ex = function() {
  for(var i = 0; i < this.nuitees.length; ++i) {
    emit(0,this.nuitees[i].nuits);
  }
}


red_ex = function (key, values) {
  var sum = 0;
  for(var i=0; i<values.length; i++)
    sum += values[i];

  return sum;
}
```

1. Expliquer ce que calcule la requête Map/Reduce donnée en exemple.

2. Donner une nouvelle version de la fonction `map_ex` afin de calculer le total des factures datées du 3 janvier 2017. La fonction `reduce_ex` ne changera pas pour ce calcul.

3. Donner des fonctions `map` et `reduce` donnant la liste des chambres ayant le prix le moins élevé (montant hors taxe par nuité) à la date du 3 janvier 2017. La fonction `reduce` doit fonctionner en cas de *re-reduce*. Il est possible de renvoyer un résultat plus riche (*i.e.* avec plus d'informations) que le résultat demandé.

   Fonction `map` :

Fonction `reduce` :