Bases de Données : exercices corrigés

Maude Manouvrier

Attention, ce document peut comporter des erreurs, à utiliser donc en connaissance de cause!!

Passage au relationnel

Exercice 1

Enoncé de l'exercice

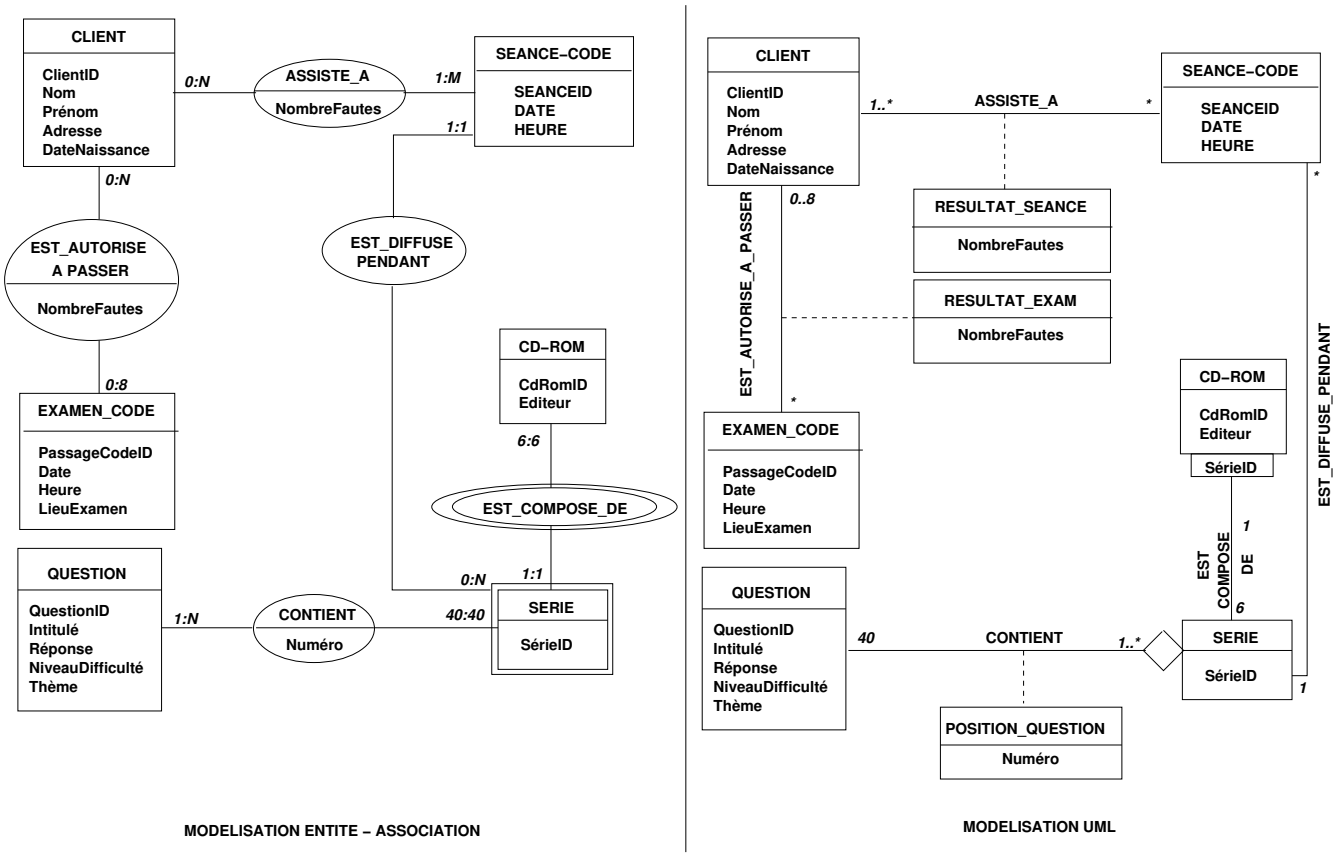

Figure 1: Modélisation E/A et UML de la base de données d'une l'auto-école.

Une auto-école souhaite construire une base de données pour gérer les examens théoriques du code de la route de ses élèves. Chaque élève est identifié par un numéro unique et est caractérisé par un nom, un prénom, une adresse et une date de naissance. Chaque élève assiste à plusieurs séances de code (autant qu'il le souhaite). Chaque séance est caractérisée par une date et une heure. A chaque séance de code, le directeur de l'auto-école choisit une série de questions sur un CD-ROM. Chaque CD-ROM est identifié par un numéro et est caractérisé par un nom d'éditeur. Chaque CD-ROM est composé de 6 séries, numérotées de 1 à 6. Chaque série est composée de 40 questions. Chaque question est identifiée par un intitulé et est caractérisée par une réponse, un niveau de difficulté et un thème. Une même question peut apparaître dans plusieurs séries avec un numéro d'ordre pour chaque série ; par exemple une même question peut apparaître comme question N°2 dans la série 5 du CD-ROM 15 et comme question N°12 dans la série 3 du CD-ROM 4. Une même série peut être projetée plusieurs fois à des séances différentes. Lorsqu'un élève assiste à une séance, il obtient le nombre de fautes (une note sur 40) qu'il a fait pour la série passée pendant la séance.

Lorsqu'un élève a obtenu, au cours des quatre dernières séances auxquelles il a assistées, un nombre de fautes inférieur ou égal à 5, le directeur de l'auto-école l'autorise à passer l'examen théorique du code de la route à une date donnée (un seul examen pour une date donnée). L'auto-école ne peut présenter que 8 élèves maximum à chaque date d'examen. Les élèves ayant obtenu plus de 5 fautes à l'examen sont recalés et doivent assister de nouveau à des séances de code avant de pouvoir se représenter à l'examen.
La base de données doit permettre de répondre à des requêtes telles que "Quel est le nombre moyen de fautes pour la série 5 du CD-ROM 14?", "Quels élèves peuvent se présenter au prochain examen du code de la route ?", "Quels élèves ont échoué au moins une fois à l'examen ?" etc.

La figure 1 présente la modélisation Entité/Association (format Merise) et la modélisation UML de l'énoncé. Les explications ne sont données que pour le schéma E/A mais peuvent être adaptées au schéma UML. Pour une compréhension de la différence entre une modélisation E/A ou UML et le passage au relationnel, vous pouvez vous reporter à l'ouvrage [4]. Les schémas de modélisation ci-avant sont sémantiquement clairs. Néanmoins, quels points nécessitent d'être précisés.

- L'ensemble d'entités *Serie* est un ensemble d'entités faibles de *CD-ROM*, au format Merise (ou une association qualifiée en UML). En effet, ce choix de modélisation a été fait pour représenter le fait que le numéro d'une série est relatif au CD-ROM auquel la série appartient.
- Les cardinalités de l'association entre les ensembles d'entités *Serie* et *CD-ROM* sont *1:1-6:6*, car une série appartient à un unique CD-ROM et un CR-ROM contient exactement 6 séries de questions. Le principe est le même pour les cardinalités de l'association entre *Serie* et *Question* : une série contient exactement 40 questions (cardinalité 40 : 40). En revanche, une même question peut apparaître dans plusieurs séries avec un numéro d'ordre différent à chaque fois, d'où la cardinalité 1 : N et l'attribut *Numéro* qui caractérise l'assocation *contient*.
- L'attribut *NombreFautes* est un attribut de l'association entre les ensembles d'entités *Client* et *Examen_ Code* et de l'association entre les ensembles d'entités *Client* et *Seance_ Code*. En effet, cet attribut caractérise l'association et non pas un client, une séance de code ou encore un examen de code. Il caractérise le lien entre deux entités de ces ensembles.

Déduisez le schéma relationnel de la base de données correspondante.

Vous préciserez les clés primaires des relations en les soulignant ainsi que les clés étrangères en les signalant par un # et en précisant à quoi elles font référence.
Dans votre schéma relationnel, chaque relation doit être spécifiée de la manière suivante :
$Nom(\underline{att_1}, \ldots, att_n)$ où Nom est le nom de la relation et $att_1, \ldots, att_n$ sont des noms d'attributs.
Le nom de la relation doit obligatoirement avoir un lien avec les noms des ensembles d'entités (classes) ou des associations du schéma de modélisation de la question 1.
Vous donnerez des explications claires et concises du passage au relationnel. Vous préciserez notamment pourquoi et comment vous créez ou modifiez certaines relations (1 ligne maximum par relation).

Correction de l'exercice 1

Le modèle relationnel déduit de la modélisation ci-dessus est le suivant. Les clés primaires sont soulignées. Les clés étrangères sont précédées d'un '#'. Pour un rappel de ces notions, vous pouvez vous référer aux pages 56 à 60 de [2]. Le passage d'un schéma de modélisation à un modèle relationnel est rappelé aux pages 120 à 143 de [4].

Client(ClientID, Nom, Prénom, Adresse, DateNaissance)
Cette relation est déduite du passage au relationnel de l'ensemble d'entités (ou classe) *Client*.

CD-ROM(CdRomID, Editeur)
Cette relation est déduite du passage au relationnel de l'ensemble d'entités (ou classe) *CD-ROM*.

Question(QuestionID, Intitulé, Réponse, NiveauDifficulté, Thème)
Cette relation est déduite du passage au relationnel de l'ensemble d'entités (ou classe) *Question*.

Serie(SerieID, #CdRomID)
Cette relation est issue du passage au relationnel de l'ensemble d'entités (classes) *Serie*. En modélisation E/A, l'ensemble d'entités *Serie* étant un ensemble d'entités faibles de *CR-ROM*, la clé primaire de la relation *Serie* est composée de l'identificateur de la série et de l'identificateur du CD-ROM auquel la série appartient.

ContenuSerie(#QuestionID, SerieID, #CdRomID, Numéro)
Cette relation est déduite du passage au relationnel de l'association *Contient*. Le couple d'attributs *#CdRomID,#SerieID* fait référence à la clé primaire de la relation *Serie*. L'attribut *#QuestionID* fait référence à la clé primaire de la relation *Question*. Il ne faut pas oublier l'attribut *Numéro* qui caractérise l'association *contient*. *Seance_ Code(SeanceID, Date, Heure, #CdRomID,#SerieID)*
Cette relation est issue du passage au relationnel de l'ensemble d'entités (classes) *Seance_ Code*. Le couple d'attributs *#CdRomID* est une clé étrangère qui fait référence à la clé primaire de la relation *Serie*. Ce couple d'attributs a été ajouté lors du passage au relationel de l'association *est_ diffusée_ pendant*, une seule série (d'un CR-ROM donné) étant diffusée pendant une séance de code (cardinalité *1:1*).

Participation(#ClientID, #SeanceID, NombreFautes)
Cette relation est issue du passage au relationnel de l'association *assiste_ à*. Les attributs *#ClientID* et *#SeanceID* sont des clés étrangères qui font respectivement références aux clés primaires des relations *Client* et *Seance*. En effet, un client peut assister à plusieurs séances et, lors d'une séance, il y a plusieurs clients. Il ne faut pas oublier l'attribut *NombreFautes* qui caractérise l'association *assiste_ à*.

Examen_ Code(PassageCodeID, Date, Heure, LieuExamen)
Cette relation est déduite du passage au relationnel de l'ensemble d'entités (ou classe) *Examen_ Code*.

PassageCode(#PassageCodeID, #ClientID, NombreFautes)
Cette relation est déduite du passage au relationnel de l'association *est_ autorisé_ à_ passer*. Les attributs *#PassageCodeID* et *#ClientID* font référence aux clés primaires des relations *Examen-Code* et *Client*.

Exercice 2

Enoncé de l'exercice

On souhaite construire une base de données gérant des revues et les articles de ces revues. Une revue est caractérisée par un nom et une périodicité. Chaque revue parait sous la forme de numéros, chaque numéro étant identifié par un nombre relatif à la revue et à l'année en cours (ex. le numéro N°12 de *Linux Magazine* en 2003 est différent du numéro N°12 de *Linux Magazine* en 2004). Un numéro est également caractérisé par un nombre de pages. Chaque numéro contient des articles écrits par un ou plusieurs auteurs. Un auteur est caractérisé par un nom, un prénom, ainsi qu'un email. Chaque article possède un titre et un contenu. Un même article peut apparaître dans plusieurs plusieurs numéros d'une même revue ou de différentes revues. Lorsqu'un article apparaît dans un numéro d'une revue, il a une page de début et une page de fin. Un article peut faire référence à d'autres articles, en précisant le numéro et la revue dans lesquels l'article référencé a été publié.

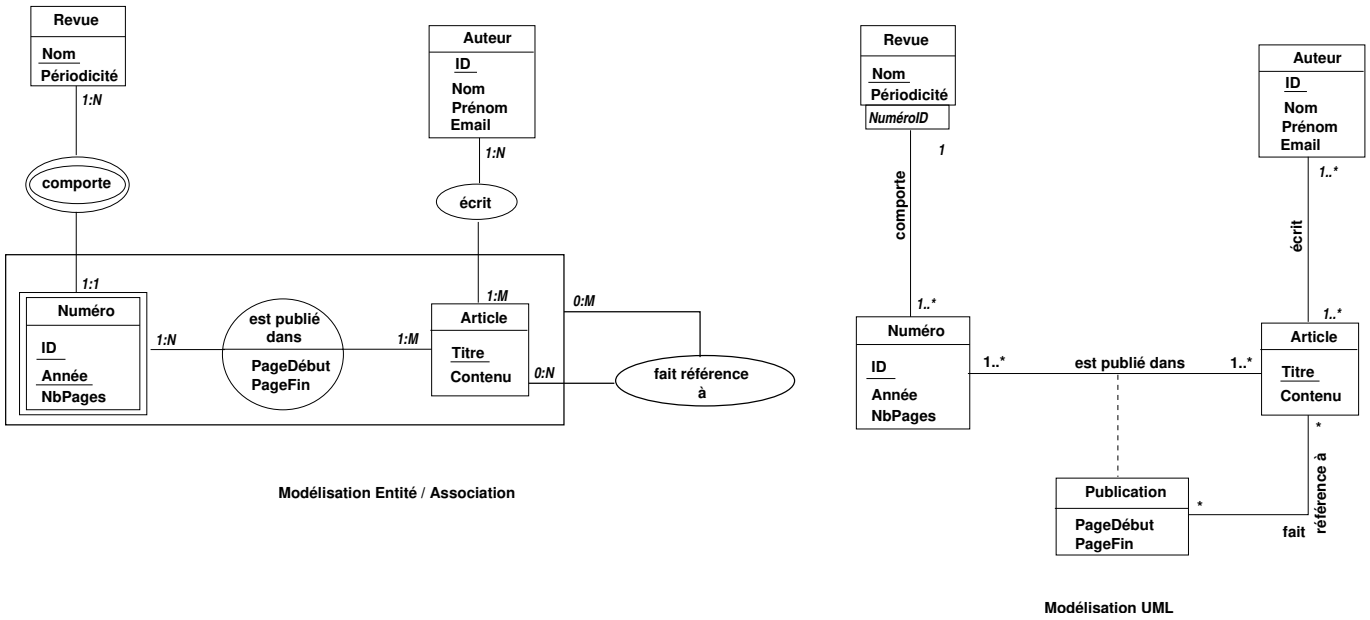

Figure 2: Modélisation E/A et UML d'une base de données gérant des revues.

La base de données doit permettre de répondre à des requêtes telles que "Combien de numéros de *Linux Magazine* sont parus en 2004 ?", "Quels sont les titres des articles parus dans au moins deux revues différentes ?", "Quels sont les auteurs ayant publiés dans le numéro 3 de la revue *L'Histoire* en 2004 ?" etc.

La figure 2 présente la modélisation Entité/Association (format Merise) et la modélisation UML de l'énoncé. Les explications ne sont données que pour le schéma E/A mais peuvent être adaptées au schéma UML. Pour une compréhension de la différence entre une modélisation E/A ou UML et le passage au relationnel, vous pouvez vous reporter à l'ouvrage [4]. Les schémas de modélisation ci-avant sont sémantiquement clairs. Néanmoins, quels points nécessitent d'être précisés.

- L'ensemble d'entités *Numéro* est un ensemble d'entités faibles de *Revue* au format Merise (ou une association qualifiée en UML). En effet, ce choix de modélisation a été fait pour représenter le fait que l'identificateur d'un numéro est relatif à la revue à laquelle le numéro appartient. Un numéro d'une revue donnée étant identifié par un nombre et une année, ces deux attributs (*ID* et *Année*) sont soulignés.

- Les cardinalités de l'association entre les ensembles d'entités *Numéro* et *Article* sont *1:N-1:M*, car un article peut apparaître dans plusieurs numéros et un numéro contient plusieurs articles. Le principe est le même pour les cardinalités de l'association entre *Auteur* et *Article*.

- Les attributs *PageDébut* et *PageFin* caractérisent l'association entre *Article* et *Numéro* (un numéro étant relatif à une revue, il est inutile de faire une association avec *Revue*). En effet, la page de début et la page de fin peuvent varier, pour un même article, lorsqu'il paraît dans plusieurs numéros différents.

- Un article peut faire référence à un autre article. Le numéro et la revue dans lesquels l'article référencé apparaît doivent être précisés dans l'article référençant. Par exemple, l'article intitulé "Correction d'exercices en bases de données" peut faire référence à l'article "Concepts généraux en BD relationnelle" du numéro 12 de l'année 2004 de la revue "Informatique magazine". A cet effet, les ensembles d'entités *Numéro* et *Article* ont été regroupés au sein d'un agrégat au format Merise (ou d'une classe-association en UML). Cet agrégat (ou cette classe-association) représente l'article référencé, c'est-à-dire l'article et le numéro de la revue où il a été publié (il est inutile d'ajouter la revue à l'agrégat puisque *Numéro* est un ensemble d'entités faibles de *Revue*). Cet agrégrat (ou cette classe-association) est associée à *Article*,

c'est-à-dire à l'article référençant, dont on ne précise pas le numéro et la revue. Les cardinalités ont pour borne inférieure 0 car un article peut ne référencer aucun autre article et un article peut ne jamais être référencé. Pour plus de détails sur l'agrégation, vous pouvez vous référer à la page 37 de l'ouvrage [2] ou aux pages 55-56 de [3].

Déduisez le schéma relationnel de la base de données correspondante.
Vous préciserez les clés primaires des relations en les soulignant ainsi que les clés étrangères en les signalant par un # et en précisant à quoi elles font référence.
Dans votre schéma relationnel, chaque relation doit être spécifiée de la manière suivante :
R_{Nom} ($\underline{att_1}$, . . . , att_n) où R_{Nom} est le nom de la relation et att_1 , . . . , att_n sont des noms d'attributs.
Le nom de la relation doit obligatoirement avoir un lien avec les noms des ensembles d'entités (classes) ou des associations du schéma de modélisation de la question 1.
Vous donnerez des explications claires et concises du passage au relationnel. Vous préciserez notamment pourquoi et comment vous créez ou modifiez certaines relations (1 ligne maximum par relation).

Correction de l'exercice 2

Le modèle relationnel déduit de la modélisation ci-dessus est le suivant. Les clés primaires sont soulignées. Les clés étrangères sont précédées d'un '#'. Pour un rappel de ces notions, vous pouvez vous référer aux pages 56 à 60 de [2]. Le passage d'un schéma de modélisation à un modèle relationnel est rappelé aux pages 120 à 143 de [4].

Revue(Nom, Périodicité)
Cette relation est déduite du passage au relationnel de l'ensemble d'entités (ou classe) *Revue*.

Numéro(ID, Année, #NomRevue, NbPages)
Cette relation est issue du passage au relationnel de l'ensemble d'entités (classes) *Numéro*, qui est un ensemble d'entités faibles (ou une association qualifiée) de *Revue*. La clé primaire de la relation *Numéro* est donc composée du couple (*ID, Année*) qui identifie un numéro pour une revue donnée et de l'attribut *#NomRevue*, clé étrangère faisant référence à la clé primaire de la relation *Revue* (c'est-à-dire faisant référence à la revue à laquelle le numéro est relatif).

Auteur(ID, Nom, Prénom, Email)
Cette relation est déduite du passage au relationnel de l'ensemble d'entités (ou classe) *Auteur*.

Article(Titre, Contenu)
Cette relation est déduite du passage au relationnel de l'ensemble d'entités (ou classe) *Article*.

Ecriture(#Titre, #IDAuteur)
Cette relation est déduite du passage au relationnel de l'association entre *Article* et *Auteur*. En effet, un article peut être écrit par plusieurs auteurs et un auteur peut écrire plusieurs articles. L'attribut *#Titre* est une clé étrangère qui fait référence à la clé primaire de la relation *Titre*. L'attribut *#IDAuteur* est une clé étrangère qui fait référence à la clé primaire de la relation *Auteur*.

Publication(#Titre, #NomRevue, #IDNuméro, #AnnéeNuméro, PageDébut, PageFin)
Cette relation est déduite du passage au relationnel de l'association entre *Article* et *Numéros*. La clé primaire de la relation est composée du titre de l'article publié (identifié par *#Titre*) et du numéro dans lequel l'article apparaît (identifié par le triplet *#NomRevue, #IDNuméro, #AnnéeNuméro*). L'attribut *#Titre* est une clé étrangère qui fait référence à la clé primaire de la relation *Titre*. Les attributs (*#NomRevue, #IDNuméro, #AnnéeNuméro*) forment une clé étrangère qui fait référence à la clé primaire de la relation *Numéro*. Il faut bien noter, ici, qu'une clé étrangère fait toujours référence à la clé primaire d'une autre relation. Or, dans cet exemple, la clé primaire de la relation *Numéro* est composée de trois attributs qui doivent également apparaître dans toutes clés étrangères y faisant référence. Il ne faut pas

non plus oublier d'ajouter dans la relation *Publication* les attributs *PageDébut* et *PageFin* qui caractérise l'association *est_publié_dans*.

Référence(#TitreArticleRéférençant, #TitreArticleRéférencé, #NomRevueArticleRéférencé, #IDNuméroArticleRéférencé, #AnnéeNuméroArticleRéférencé)
Cette relation est déduite du passage au relationnel de l'association entre l'ensemble d'entités (ou la classe) *Article* et l'agrégat (ou la classe-association) regroupant *Numéro* et *Article*. Les noms des attributs sont particulièrement longs pour que la sémantique soit claire. L'attribut *#TitreArticleRéférençant* est une clé étrangère qui fait référence à la clé primaire de la relation *Article*. Cet attribut représente le titre de l'article référençant (contenant une référence à un article publié dans un numéro). L'attribut *#TitreArticleRéférencé* est une clé étrangère qui fait référence à la clé primaire de la relation *Article*. Cet attribut représente le titre de l'article référencé. Les attributs (*#NomRevueArticleRéférencé, #IDNuméroArticleRéférencé, #AnnéeNuméroArticleRéférencé*) forment une clé étrangère qui fait référence à la clé primaire de la relation *Numéro*. Ils représentent le numéro de la revue dans lequel a été publié l'article référencé.

Langage d'interrogation

Exercice 3

Enoncé de l'exercice

On suppose qu'une bibliothèque gère une base de données dont le schéma est le suivant (les clés primaires des relations sont soulignées) :

Emprunt(Personne, Livre, DateEmprunt, DateRetourPrevue, DateRetourEffective)
Retard(Personne, Livre, DateEmprunt, PenalitéRetard)

Exprimer, lorsque cela est possible, les requêtes suivantes en algèbre relationnelle, en calcul à variable nuplet et en SQL.

1. Quelles sont les personnes ayant emprunté le livre "Recueil Examens BD" ?
2. Quelles sont les personnes n'ayant jamais rendu de livre en retard ?
3. Quelles sont les personnes ayant emprunté tous les livres (empruntés au moins une fois) ?
4. Quels sont les livres ayant été empruntés par tout le monde (i.e. tous les emprunteurs) ?
5. Quelles sont les personnes ayant toujours rendu en retard les livres qu'elles ont empruntés ?

Correction de l'exercice 3

Dans cet exercice, le schéma relationnel est particulièrement simple, afin que l'expression des requêtes soit facile à exprimer. Il s'agit néanmoins de requêtes complexes. Vous pouvez vous entraîner à exprimer ces requêtes en améliorant le schéma, c'est-à-dire en ajoutant deux relations *Personne* et *Livre* et précisant les clés étrangères dans les relations *Emprunt* et *Retard* faisant référence à une personne et à un livre.

1. **Quelles sont les personnes ayant emprunté le livre "Recueil Examens BD" ?**

   En algèbre relationnelle : $\Pi_{Personne}(\sigma_{Livre='Recueil...'}(Emprunt))$

   L'algèbre relationnelle est un langage composé d'opérations ensemblistes. Il permet d'indiquer comment le résultat de la requête est calculé en termes d'opérations ensemblistes sur des ensembles de nuplets (les relations). Dans cette requête par exemple, le résultat est calculé en parcourant tous les nuplets de la relation *Emprunt*, en y sélectionnant les nuplets dont l'attribut *Livre* a pour valeur *'Recueil...'* et en prenant uniquement les valeurs de l'attribut *Personne* (i.e. en projetant sur l'attribut *Personne*).

   En calcul relationnel : $\{t.Personne \mid Emprunt(t) \wedge (u.Livre =' Recueil...') \}$

   Le calcul relationnel décrit, sous forme logique, le résultat de la requête (sans préciser comment on le calcule). Le résultat de la requête contient les valeurs de l'attribut *Personne* des nuplets t de la relation *Emprunt* tels que l'attribut *Livre* corresponde à 'Recueil Examens

BD'.

En SQL:

```
SELECT Personne
 FROM Emprunt WHERE Livre = 'Recueil...'
```

Il aurait également été possible de remplacer la clause `WHERE` par `WHERE Livre LIKE 'Recueil%'` indiquant que l'on recherche les emprunteurs des ouvrages dont le titre commence par 'Recueil'.

2. **Quelles sont les personnes n'ayant jamais rendu de livre en retard ?**

   En algèbre relationnelle : $\Pi_{Personne}(Emprunt) - \Pi_{Personne}(Retard)$

   La résultat de la requête est calculé en prenant toutes les valeurs de l'attribut *Personne* dans la relation *Emprunt* et en éliminant les valeurs de ce même attribut apparaissant également dans la relation *Retard*. Il s'agit d'une différence entre deux ensembles.

   En calcul relationnel :

   $\{t.Personne \mid Emprunt(t) \wedge \neg[\exists\ u\ Retard(u) \wedge (u.Personne = t.Personne)\ )]\}$

   Le résultat de la requête contient les valeurs de l'attribut *Personne* des nuplets t de la relation *Emprunt* (donc des personnes empruntant) tels qu'il n'existe pas de nuplets u dans la relation *Retard* avec la même valeur pour l'attribut *Personne* (donc telles qu'il n'existe pas de retards associés à ces personnes).

   En SQL, deux manières possibles, par simple traduction en SQL de la requête en calcul relationnel (le calcul relationnel étant à l'origine de la syntaxe de SQL) :

```
SELECT t.Personne FROM Emprunt t
  WHERE NOT EXISTS (SELECT *  FROM Retard u
                      WHERE u.Personne=t.Personne
                    )
```

```
SELECT Personne FROM Emprunt
  WHERE Personne NOT IN
        (SELECT Personne FROM Retard)
```

   Les variables nuplet (ex. t et u) ne sont nécessaire que lorsqu'il y a ambiguïté au niveau des noms d'attributs (cf. requête de gauche).

3. **Quelles sont les personnes ayant emprunté tous les livres (empruntés au moins une fois) ?**

   En algèbre relationnelle : $\Pi_{Personne,Livre}(Emprunt) \div \Pi_{Livre}(Emprunt)$
   Le résultat de cette requête est calculé en utilisant l'opérateur de division. Pour une bonne compréhension de la division, vous pouvez vous reporter à la page 99 de [2].
   La sous-requête $\Pi_{Livre}(Emprunt)$ correspond à la liste des livres empruntés. Le résultat de la sous-requête $\Pi_{Personne,Livre}(Emprunt)$ contient tous les couples *(Personne, Livre emprunté au moins une fois par cette personne)*. Le résultat de la division sera donc la liste des personnes associées, dans le résultat de $\Pi_{Personne,Livre}(Emprunt)$, à chacun des livres apparaissant dans le résultat de la requête $\Pi_{Livre}(Emprunt)$.

En calcul relationnel :

$\{t.Personne \mid Emprunt(t) \wedge [\forall\ u\ (Emprunt(u)) \Longrightarrow (\exists\ v\ Emprunt(v) \wedge (v.Personne = t.Personne) \wedge (u.Livre = v.Livre)\ )]\}$
Le résultat de la requête contient les valeurs de l'attribut *Personne* des nuplets t de la relation *Emprunt* tels que quel que soit un nuplet s'il s'agit d'un livre emprunté (donc d'un nuplet u dans *Emprunt*) alors on trouve un nuplet v dans *Emprunt* associant cette personne à ce livre (c'est-à-dire $v.Personne = t.Personne$ et $u.Livre = v.Livre$).

On peut également l'écrire de la manière suivante :
$\{t.Personne \mid Emprunt(t) \wedge [\forall\ u\ \neg(Emprunt(u)) \vee (\exists\ v\ Emprunt(v) \wedge (v.Personne = t.Personne) \wedge (u.Livre = v.Livre)\ )]\}$

Ce qui signifie que le résultat de la requête contient les valeurs de l'attribut *Personne* des nuplets t de la relation *Emprunt* tels que quel que soit un nuplet u soit c'est n'est pas un nuplet de *Emprunt* soit (implicitement c'est un nuplet de *Emprunt* et) on trouve un nuplet v dans *Emprunt* associant cette personne à ce livre (c'est-à-dire $v.Personne = t.Personne$ et $u.Livre = v.Livre$).

D'où dit de manière négative :
$\{t.Personne \mid Emprunt(t) \wedge \neg[\exists\, u\, Emprunt(u)\ \neg(\exists\, v\, Emprunt(v) \wedge (v.Personne = t.Personne) \wedge (u.Livre = v.Livre)\ )]\}$

En SQL, simple traduction de la requête en calcul relationnel :

```
SELECT t.Personne
 FROM Emprunt t
  WHERE NOT EXISTS ( SELECT *
                       FROM Emprunt u WHERE NOT EXISTS ( SELECT *
                                                           FROM Emprunt v
                                                            WHERE v.Personne=t.Personne
                                                              AND v.Livre=u.Livre
                                                       )
                   )
```

4. **Quels sont les livres ayant été empruntés par tout le monde (i.e. tous les emprunteurs) ?**

En algèbre relationnelle : $\Pi_{Personne,Livre}(Emprunt) \div \Pi_{Personne}(Emprunt)$

Le résultat de cette requête est calculé en utilisant également l'opérateur de division.
La sous-requête $\Pi_{Personne}(Emprunt)$ correspond à la liste des emprunteurs. Le résultat de la sous-requête $\Pi_{Personne,Livre}(Emprunt)$ contient tous les couples *(Personne ayant emprunté au moins une fois, Livre emprunté au moins une fois par cette personne)*. Le résultat de la division sera donc la liste des livres associés, dans le résultat de $\Pi_{Personne,Livre}(Emprunt)$, à chacun des emprunteurs apparaissant dans le résultat de la requête $\Pi_{Personne}(Emprunt)$.

En calcul relationnel :

$\{t.Livre \mid Emprunt(t) \wedge [\forall\ u\ (Emprunt(u)) \Longrightarrow (\exists\ v\ Emprunt(v) \wedge (v.Livre = t.Livre) \wedge (v.Personne = u.Personne)\ )]\}$

Le résultat de la requête contient les valeurs de l'attribut *Livre* des nuplets t de la relation *Emprunt* tels que quel que soit un nuplet u, s'il s'agit d'un emprunteur (donc d'un nuplet u dans *Emprunt*) alors on trouve un nuplet v dans *Emprunt* associant ce livre à cet

emprunteur (c'est-à-dire $v.Livre = t.Livre$ et $v.Personne = u.Personne$).

On peut également l'écrire de la manière suivante :
$\{t.Livre \mid Emprunt(t) \wedge [\forall\ u\ \neg(Emprunt(u)) \vee (\exists\ v\ Emprunt(v) \wedge (v.Livre = t.Livre) \wedge (v.Personne = u.Personne)\ )]\}$

Ce qui signifie que le résultat de la requête contient les valeurs de l'attribut *Livre* des nuplets t de la relation *Emprunt* tels que quel que soit un nuplet u, soit il ne s'agit pas d'un nuplet u dans *Emprunt*, soit (il s'agit d'un d'un nuplet u dans *Emprunt* et) il existe un nuplet v dans *Emprunt* associant ce livre à cet emprunteur (c'est-à-dire $v.Livre = t.Livre$ et $t.Personne = u.Personne$). D'où dit de manière négative :
$\{t.Livre \mid Emprunt(t) \wedge \neg[\exists\ u\ Emprunt(u)\ \neg(\exists\ v\ Emprunt(v) \wedge (v.Livre = t.Livre) \wedge (v.Personne = u.Personne)\ )]\}$

En SQL, simple traduction de la requête en calcul relationnel :

```
SELECT t.Livre FROM Emprunt t
  WHERE NOT EXISTS ( SELECT * FROM Emprunt u
                       WHERE NOT EXISTS ( SELECT * FROM Emprunt v
                                            WHERE v.Livre=t.Livre AND v.Personne=u.Personne
                                        )
                   )
```

5. **Quelles sont les personnes ayant toujours rendu en retard les livres qu'elles ont empruntés ?**

En algèbre relationnelle : Il n'est pas possible d'exprimer cette requête par une division. La requête est donc décomposée en deux sous-requêtes. La requête, R_1, ci-dessous, retourne la liste des personnes ayant emprunté au moins un livre sans le rendre en retard.

$$R_1 = \Pi_{Personne}\,[\Pi_{Personne,Livre,DateEmprunt}(Emprunt) - \Pi_{Personne,Livre,DateEmprunt}(Retard)]$$

La requête ci-dessous enlève de la liste des personnes qui empruntent des livres (sous-requête de gauche) la liste des personnes ayant rendu au moins un livre sans retard (requête R_1). Cela correspond à comment calculer le résultat de la requête que l'on recherche.

$$\Pi_{Personne}(Emprunt) - R_1$$

En calcul relationnel :

$\{t.Personne \mid Emprunt(t) \wedge [\forall\ u\ [Emprunt(u) \wedge (u.Personne = t.Personne)] \Longrightarrow (\exists\ v\ Retard(v) \wedge (v.Personne = u.Personne) \wedge (u.Livre = v.Livre)\ )]\}$

Le résultat de la requête contient les valeurs de l'attribut *Personne* des nuplets t de la relation *Emprunt* tels que quel que soit un nuplet s'il s'agit d'un livre emprunté par cette personne (donc d'un nuplet u dans *Emprunt* tel que $u.Personne = t.Personne$) alors on trouve un nuplet v dans *Retard* associant cette personne à ce livre (c'est-à-dire $v.Personne = u.Personne$ et $u.Livre = v.Livre$).

On peut également écire :
$\{t.Personne \mid Emprunt(t) \wedge [\forall\ u\ \neg[Emprunt(u) \wedge (u.Personne = t.Personne)] \vee (\exists\ v\ Retard(v) \wedge (v.Personne = u.Personne) \wedge (u.Livre = v.Livre)\ )]\}$

Le résultat de la requête contient les valeurs de l'attribut *Personne* des nuplets t de la relation *Emprunt* tels que quel que soit un nuplet soit il ne s'agit pas d'un livre emprunté par cette personne (donc d'un nuplet u dans *Emprunt* tel que $u.Personne = t.Personne$)

soit on trouve un nuplet v dans *Retard* associant cette personne à ce livre (c'est-à-dire $v.Personne = u.Personne$ et $u.Livre = v.Livre$).

D'où dit de manière négative :
$\{t.Personne \mid Emprunt(t) \wedge \neg[\exists\, u\ Emprunt(u) \wedge (u.Personne = t.personne)\ \neg(\exists\, v\ Retard(v) \wedge (v.Personne = u.Personne) \wedge (u.Livre = v.Livre)\ )]\}$

En SQL, là encore , simple traduction de la requête en calcul relationnel:

```
SELECT t.Personne
 FROM Emprunt t
  WHERE NOT EXISTS (SELECT * FROM Emprunt u WHERE u.Personne=t.Personne
                      AND NOT EXISTS (SELECT * FROM Retard v WHERE v.Personne=u.Personne
                                        AND v.Livre=u.Livre )
                    )
```

Exercice 4

Enoncé de l'exercice

Un organisme de gestion de spectacles, de salles de concert et de vente de billets de spectacles gère une base de données dont le schéma relationnel est le suivant :

Spectacle(Spectacle_ID, Titre, DateDéb, Durée, Salle_ID, Chanteur)
Concert (Concert_ID, Date, Heure, Spectacle_ID)
Salle (Salle_ID, Nom, Adresse, Capacité)
Billet (Billet_ID, Concert_ID, Num_Place, Catégorie, Prix)
Vente (Vente_ID, Date_Vente, Billet_ID, MoyenPaiement)

Les attributs soulignés sont les attributs appartenant à la clé primaire. Ils sont de type entier. L'attribut *Salle_ID* de la relation *Spectacle* est une clé étrangère qui fait référence à l'attribut de même nom de la relation *Salle*. L'attribut *Spectacle_ID* de la relation *Concert* est une clé étrangère qui fait référence à l'attribut de même nom de la relation *Spectacle*. L'attribut *Concert_ID* de la relation *Billet* est une clé étrangère qui fait référence à l'attribut de même nom de la relation *Concert*. L'attribut *Billet_ID* de la relation *Vente* est une clé étrangère qui fait référence à l'attribut de même nom de la relation *Billet*.

Exprimez, lorsque cela est possible, les requêtes suivantes en algèbre relationnelle, en calcul relationnel à variable nuplet et en SQL.

1. Quelles sont les dates du concert de Corneille au Zenith ?
2. Quels sont les noms des salles ayant la plus grande capacité ?
3. Quels sont les chanteurs n'ayant jamais réalisé de concert à la Cygale ?
4. Quels sont les chanteurs ayant réalisé au moins un concert dans toutes les salles ?
5. Quels sont les dates et les identificateurs des concerts pour lesquels il ne reste aucun billet invendu ?

Correction de l'exercice 4

1. **Quelles sont les dates du concert de Corneille au Zenith ?**

   En algèbre relationnelle : $\Pi_{Date}[Concert \bowtie \sigma_{Chanteur='Corneille'}(Spectacle) \bowtie \sigma_{Nom='Zenith'}(Salle)]$

Cette requête comporte deux jointures naturelles. La première jointure, entre les relations *Concert* et *Spectacle*, associe les nuplets de *Spectacle*, correspondant aux spectacles du chanteur 'Corneille' (puisqu'il y a une sélection avant), avec les nuplets de la relation *Concert* ayant la même valeur pour l'attribut *Spectacle_ID*. La jointure se fait naturellement sur l'attribut de même nom, *Spectacle_ID*. La deuxième jointure associe les nuplets résultats de la première jointure (donc les concerts des spectacles de 'Corneille') avec le nuplets correspondant à la salle du 'Zenith' (résultat de la requête de sélection $\sigma_{Nom='Zenith'}(Salle)$). La jointure se fait naturellement sur l'attribut commun *Salle_ID*. La projection finale se fait sur l'attribut *Date*.

En calcul relationnel :

$\{t.Date \mid Concert(t) \wedge [\exists\, u, v\ Spectacle(u) \wedge Salle(v) \wedge (u.Spectacle\_ID = t.Spectacle\_ID) \wedge (u.Chanteur =' Corneille') \wedge (v.Nom =' Zenith') \wedge (u.Salle\_ID = v.Salle\_ID)\ ]\ \}$

La requête retourne les dates des concerts pour lesquels il existe un spectacle de 'Corneille' associé à la salle du 'Zenith'. Le résultat de la requête contient donc les valeurs de l'attribut *Date* des nuplets t de la relation *Concert* tels qu'il existe un nuplet u dans *Spectacle*, correspondant à un spectacle de 'Corneille' (c'est-à-dire dont l'attribut *Chanteur* a pour valeur 'Corneille'), avec la même valeur pour l'attribut *Spectacle_ID* que le nuplet t et tels qu'il existe aussi un nuplet v dans la relation *Salle*, correspondant à la salle du 'Zenith' (dont l'attribut *Nom* a pour valeur 'Zenith'), avec la même valeur pour l'attribut *Salle_ID* que celle de l'attribut *Salle_ID* du nuplet u.

En SQL, par traduction immédiate de la requête en calcul à variable nuplet :

```
SELECT Date
 FROM Concert t, Spectacle u, Salle v
  WHERE t.Spectacle_ID = u.Spectacle_ID
   AND u.Chanteur = 'Corneille'
    AND u.Salle_ID = v.Salle_ID
     AND v.Nom = 'Zenith'
```

2. **Quels sont les noms des salles ayant la plus grande capacité ?**

En algèbre relationnelle : Cette requête ne peut pas s'écrire en algèbre relationnelle non étendue. Il faut un opérateur maximum. Pour plus de détails sur l'algèbre relationnelle étendue, vous pouvez vous reporter aux pages 103 à 111 de [3] ou aux pages 221 à 230 de [1].

En algèbre relationnelle étendue[1], la requête s'exprime par:
$\Pi_{Nom}(\sigma_{(Capacite>=CapaciteMax)}\Pi_{Salle\_ID,CapacitMax}[Salle\times(\Pi_{MAX(Capacite)\longrightarrow CapaciteMax)}(Salle))])$

La requête $\Pi_{Salle\_ID,CapacitMax}[Salle\times(\Pi_{MAX(Capacite)\longrightarrow CapaciteMax)}(Salle))]$ retourne une relation temporaire de deux colonnes, la pemière contenant les valeurs de l'attribut *Salle_ID* de la relation *Salle* et la deuxième colonne contenant une seule valeur (repétée pour toutes les valeurs de *Salle_ID*) correspondant à la valeur maximale de l'attribut *Capacité* (calculée par la fonction d'agrégation *MAX* et renommée en *CapaciteMAX*). L'opérateur utilisé est le produit cartésien ($\times$). Pour obtenir le nom des salles avec la plus grande capacité, il suffit donc de joindre à cette relation temporaire à la relation *Salle* et de sélectionner les nuplets ayant une valeur de *Capacité* superieure ou égale à celle de l'attribut *CapaciteMax*.

En calcul relationnel :
$\{t.Nom \mid Salle(t) \wedge \neg[\exists\ u\ Salle(u)\ (u.Capacite >= t.Capacite)\ ]\ \}$

Cette requête retourne les valeurs de l'attribut *Nom* des nuplets t de la relation *Salle* pour

[1] **L'algèbre relationnelle étendue n'est pas au programme de l'examen.**

lesquels il n'existe pas de nuplets u dans *Salle* avec une valeur de l'attribut *Capacité* supérieure ou égale.

En SQL:

Il est possible de traduire directement la requête exprimée en calcul relationnel, comme ci-dessous.

```
SELECT Nom
 FROM Salle t
  WHERE NOT EXISTS (SELECT *
                     FROM Salle u
                      WHERE u.Capacité >= t. Capacité)
```

Il est également possible d'utiliser l'opérateur d'agrégation MAX, comme pour la requête suivante.

```
SELECT Nom
 FROM Salle
  WHERE Capacité >= ( SELECT (MAX(Capacité)
                        FROM Salle
                     )
```

Il est également possible d'utiliser le mot-clé `ALL` :

```
SELECT Nom
 FROM Salle
  WHERE Capacité >= ALL ( SELECT Capacité
                            FROM Salle
                         )
```

3. **Quels sont les chanteurs n'ayant jamais réalisé de concert à la Cygale ?**

En algèbre relationnelle : $\Pi_{Chanteur}(Spectacle) - \Pi_{Chanteur}[Spectacle \bowtie \sigma_{(Nom='Cygale')}(Salle)]$

La requête $\Pi_{Chanteur}[Spectacle \bowtie \sigma_{(Nom='Cygale')}(Salle)]$ retourne les chanteurs ayant chanté au moins une fois dans la salle de la 'Cygale'. Le résultat de la requête fnale est obtenu en supprimant ces chanteurs de la liste de tous les chanteurs.

En calcul relationnel :

$\{t.Chanteur \mid Spectacle(t) \wedge \neg[\exists\ u,\ v\ Spectacle(u) \wedge Salle(v) \wedge (v.Nom =' Cygale') \wedge (u.Chanteur = t.Chanteur) \wedge (u.Salle\_ID = v.Salle\_ID)\ ]\ \}$

La requête retourne les valeurs de l'attribut *Chanteur* des nuplets t de la relation *Spectacle* tels qu'il ne soit pas possible de trouver un spectacle de ce même chanteur à la 'Cygale' (i.e. de trouver un nuplet u dans *Spectacle* avec la même valeur pour l'attribut *Chanteur* et un nuplet v dans *Salle* avec 'Cygale' comme valeur de l'attribut *Nom* et avec la même valeur que $u.Salle\_ID$ pour l'attribut *Salle_ID*).

En SQL:

```
SELECT Chanteur
 FROM Spectacle
  WHERE Chanteur NOT IN (SELECT Chanteur
```

```
                    FROM Spectacle u, Salle v
                     WHERE u.Salle_ID=v.Salle_ID
                      AND v.Nom='Cygale'
                )
```

Cette requête peut aussi s'exprimer avec un `NOT EXISTS` en utilisant une variable nuplet t dans le premier $FROM$, par une simple traduction du calcul relationnel :

```
SELECT Chanteur
 FROM Spectacle t
  WHERE Chanteur NOT EXISTS ( SELECT *
                               FROM Spectacle u, Salle v
                                WHERE u.Salle_ID=v.Salle_ID
                                 AND v.Nom='Cygale'
                                  AND t.CHanteur=u.Chanteur
                            )
```

4. **Quels sont les chanteurs ayant réalisé au moins un concert dans toutes les salles ?**

   En algèbre relationnelle : $\Pi_{Chanteur,Salle\_ID}(Spectacle \bowtie Salle) \div \Pi_{Salle\_ID}(Salle)$

   La requête $\Pi_{Salle\_ID}(Salle)$ retourne tous les identificateurs de salle.
   La requête $\Pi_{Chanteur,Salle\_ID}(Spectacle \bowtie Salle)$ retourne une relation associant à chaque chanteur l'identificateur de la salle dans laquelle il a réalisé au moins un spectacle.
   La division va donc retourner les chanteurs associés au moins une fois à toutes les salles de la base.

   En calcul relationnel :
   $\{t.Chanteur \mid Spectacle(t) \wedge [\forall\, u\, (Salle(u)) \Longrightarrow (\exists\, v\, Spectacle(v) \wedge (v.Chanteur = t.Chanteur) \wedge (u.Salle\_ID = v.Salle\_ID)\ )\ ]\ \}$

   La requête retourne les valeurs de l'attribut *Chanteur* des nuplets t de la relation *Spectacle* tels que pour quel que soit un nuplet, s'il s'agit d'une salle (donc un nuplet u pris dans *Salle*), alors il existe un spectacle de ce chanteur dans cette salle (donc il existe un nuplet v dans *Spectacle* correspondant à ce chanteur, avec $v.Chanteur = t.Chanteur$, et à cette salle, avec $u.Salle\_ID = v.Salle\_ID$).

   On peut également écrire :
   $\{t.Chanteur \mid Spectacle(t) \wedge [\forall\, u\, \neg(Salle(u)) \vee (\exists\, v\, Spectacle(v) \wedge (v.Chanteur = t.Chanteur) \wedge (u.Salle\_ID = v.Salle\_ID)\ )\ ]\ \}$

   La requête retourne les valeurs de l'attribut *Chanteur* des nuplets t de la relation *Spectacle* tels que pour quel que soit un nuplet, soit il ne s'agit pas d'une salle (donc il ne s'agit pas d'un nuplet u de *Salle*), soit (implicitement il s'agit d'un nuplet u de *Salle* et) il existe un spectacle de ce chanteur dans cette salle (donc il existe un nuplet v dans *Spectacle* correspondant à ce chanteur, avec $v.Chanteur = t.Chanteur$, et à cette salle, avec $u.Salle\_ID = v.Salle\_ID$).

   D'où dit de manière négative :

   $\{t.Chanteur \mid Spectacle(t) \wedge \neg[\exists\, u\, Salle(u)\, \neg(\exists\, v\, Spectacle(v) \wedge (v.Chanteur = t.Chanteur) \wedge (u.Salle\_ID = v.Salle\_ID)\ )\ ]\ \}$

   En SQL:

```
SELECT Chanteur FROM Spectacle t WHERE NOT EXISTS
```

```
( SELECT * FROM Salle u WHERE NOT EXISTS
    ( SELECT * FROM Spectacle v
       WHERE v.Chanteur = t. Chanteur AND u.Salle_ID = v.Salle_ID
    )
)
```

5. **Quels sont les dates et les identificateurs des concerts pour lesquels il ne reste aucun billet invendu ?**

   En algèbre relationnelle : Cette requête étant complexe et ne peut pas s'exprimer à l'aide d'une division. Il est plus simple de l'écrire en la décomposant. Une première sous-requête R_1 va permettre de déterminer les billets invendus :

   $$R_1 = \Pi_{Billet\_ID}(Billet) - \Pi_{Billet\_ID}(Vente)$$

   La requête R_1 supprime de la liste des billets ($\Pi_{Billet\_ID}(Billet)$), ceux qui ont été vendus ($\Pi_{Billet\_ID}(Vente)$). Pour obtenir les concerts auxquels appartiennent ces billets invendus, il faut faire une jointure avec la relation *Billet* (pour obtenir la valeur de l'attribut *Concert_ID* associé au billet) puis avec *Concert* (pour obtenir la date du concert associé), soit :

   $$R_2 = \Pi_{Date,Concert\_ID}(Concert \bowtie Billet \bowtie \underbrace{[\Pi_{Billet\_ID}(Billet) - \Pi_{Billet\_ID}(Vente)]}_{R_1})$$

   Au final, on supprime la liste des identificateurs de concerts et de leur date associée au résultat de la requête R_2, soit :

   $$\Pi_{Date,Concert\_ID}(Concert) - \overbrace{\Pi_{Date,Concert\_ID}(Concert \bowtie Billet \bowtie \underbrace{[\Pi_{Billet\_ID}(Billet) - \Pi_{Billet\_ID}(Vente)]}_{R_1})}^{R_2}$$

   En calcul relationnel :

   $\{t.Concert\_ID, t.Date \mid Concert(t) \wedge [\forall\ u\ Billet(u)\ (u.Concert\_ID = t.Concert\_ID)\ (\exists\ v\ Vente(v) \wedge (v.Billet\_ID = u.Billet\_ID)\ )\ ]\ \}$

   La requête retourne les valeurs des attributs *Concert_ID* et *Date* des nuplets t de la relation *Concert* tels que pour tous les billets de ce concert (donc pour tous les nuplets u dans *Billet* tels que $u.Concert\_ID = t.Concert\_ID$), il existe une vente de ce billet (donc il existe un nuplet v dans *Vente* correspondant à ce billet, i.e. tel que $v.Billet\_ID = u.Billet\_ID$).

   D'où dit de manière négative :
   $\{t.Concert\_ID, t.Date \mid Concert(t) \wedge \neg[\exists\ u\ Billet(u)\ (u.Concert\_ID = t.Concert\_ID)$
   $\neg(\exists\ v\ Vente(v) \wedge (v.Billet\_ID = u.Billet\_ID)\ )\ ]\ \}$

   En SQL:

```
SELECT Concert_ID, Date
 FROM Concert t
  WHERE NOT EXISTS (SELECT * FROM Billet u
                     WHERE u.Concert_ID=t.Concert_ID
                      AND NOT EXISTS (SELECT * FROM Vente v
                                       WHERE u.Billet_ID = v.Billet_ID
                                     )
                   )
```

Dépendances fonctionnelles et normalisation

Exercice 5

Enoncé de l'exercice

Soit un schéma de bases de données contenant les relation suivantes :

Bureau(NumBureau, NumTelephone, Taille) avec
$F_{Bureau} = \{ NumBureau \rightarrow NumTelephone, Taille; NumTelephone \rightarrow NumBureau; \}$
Occupant(NumBureau, PersonneID) avec $F_{Occupant} = \{ NumBureau \rightarrow PersonneID \}$
Materiel(NumBureau, NumPC) avec $F_{Materiel} = \{ NumPC \rightarrow NumBureau \}$

1. Les contraintes ci-dessous sont-elles vérifiées par ce schéma de bases de données? Si la réponse est positive, expliquez pourquoi. Si la réponse est négative, indiquez quelle(s) dépendance(s) fonctionnelle(s) il faut ajouter/supprimer ou modifier pour que la contrainte soit vérifiée.

   (a) "Un bureau peut contenir plusieurs postes téléphoniques."

   (b) "Il y a une et une seule personne par bureau."

   (c) "Un bureau contient un seul ordinateur."

2. A partir des familles de dépendances fonctionnelles initiales données dans l'énoncé, indiquez quelles sont les clés minimales possibles de chaque relation.

Correction de l'exercice 5

1. **Vérification des contraintes exprimées par des dépendances fonctionnelles :**

   (a) *"Un bureau peut contenir plusieurs postes téléphoniques"*
   Cette contrainte n'est pas vérifiée car F_{Bureau} contient la dépendance fonctionnelle $NumBureau \rightarrow NumTelephone$ donc à un bureau est associé un et un seul numéro de téléphone. Pour que la contrainte soit vérifiée, il faudrait supprimer cette dépendances fonctionnelle.

   (b) *"Il y a une et une seule personne par Bureau."*
   Cette contrainte est vérifiée car $F_{Occupant}$ contient la dépendance fonctionnelle $NumBureau \rightarrow PersonneID$, donc à un numéro de bureau est associée une et une seule personne.

   (c) *"Un bureau contient un seul ordinateur."*
   Cette contrainte n'est pas vérifiée car il y a juste l'information qu'un ordinateur est dans un seul bureau ($NumPC \rightarrow NumBureau$) mais pas l'inverse. Pour que la contrainte soit vérifiée, il faudrait ajouter la dépendance fonctionnelle $NumBureau \rightarrow NumPC$.

2. **Détermination des clés minimales des relations :**

   $F_{Bureau} = \{\ NumBureau \rightarrow NumTelephone, Taille; NumTelephone \rightarrow NumBureau; \}$
   La relation *Bureau* a donc deux clés minimales possibles : $NumBureau$ et $NumTelephone$.

   En effet, à partir de l'attribut $NumBureau$ il est possible de déduire les deux autres attributs de la relation (par la première dépendance fonctionnelle). Par l'attribut $NumTelephone$, il est possible de déduire $NumBureau$ (2ème dépendance fonctionnelle) et donc l'attribut $Taille$ (par la première dépendance fonctionnelle). On a donc :
   $[NumBureau]^+ = \{NumBureau, NumTelephone, Taille\}$ car $NumBureau \rightarrow NumTelephone, Taille$.
   et $[NumTelephone]^+ = \{NumTelephone, NumBureau, Taille\}$, car $NumTelephone \rightarrow NumBureau$
   et donc par transitivité avec $NumBureau \rightarrow Taille$, on obtient $NumTelephone \rightarrow Taille$.

   $F_{Occupant} = \{\ NumBureau \rightarrow PersonneID\ \}$
   La relation *Occupant* a donc une seule clé minimale possible : $NumBureau$.

   $F_{Materiel} = \{\ NumPC \rightarrow NumBureau\ \}$
   La relation *Materiel* a donc une seule clé minimale possible : $NumPC$.

   Pour plus de détails sur les dépendances fonctionnelles, vous pouvez vous reporter aux pages 422 à 430 de [2].

Exercice 6

Enoncé de l'exercice

Soit R une relation dont le schéma est le suivant :
R(UtilisateurID, Nom, Prénom, AdresseEmail, Login, Passwd, ServeurMail).

1. Exprimer, à l'aide de dépendances fonctionnelles, les contraintes suivantes que doivent vérifier les instances de la relation R :

   (a) "On peut déduire le nom et le prénom d'un utilisateur à partir de son identificateur."

   (b) "Un utilisateur (identifié par son identificateur) possède un seul login et un seul password par serveur de mails."

   (c) "Une adresse email est associée à un et un seul identificateur d'utilisateur."
   Attention : un utilisateur peut avoir plusieurs adresses de mails.

   (d) "Une adresse email est associée à un et un seul serveur de mails."

2. Indiquer, à partir de la famille de dépendances fonctionnelles, issue de la question 1, quelles sont les clés mimimales de R.

3. Indiquer, à partir de la famille de dépendances fonctionnelles, issue de la question 1, en quelle forme normale est la relation R.

Correction de l'exercice 6

1. **Expression de contraintes par des dépendances fonctionnelles :**

   (a) *"On peut déduire le nom et le prénom d'un utilisateur à partir de son identificateur."*
   Cette contrainte s'exprime par la dépendance fonctionnelle :
   $UtilisateurID \longrightarrow Nom, Prénom$
   En effet, à un identificateur d'utilisateur est associé un et un seul nom et un et un seul prénom.

   (b) *"Un utilisateur (identifié par son identificateur) possède un seul login et un seul password par serveur de mails."*
   Cette contrainte s'exprime par la dépendance fonctionnelle :
   $UtilisateurID, ServeurMail \longrightarrow Login, Passwd$
   En effet pour un couple (*identificateur d'utilisateur, serveur de mail*) est associé un et un seul login et un et un seul mot de passe.

   (c) *"Une adresse email est associée à un et un seul identificateur d'utilisateur."*
   Attention : un utilisateur peut avoir plusieurs adresses de mails.
   Cette contrainte s'exprime par la dépendance fonctionnelle :
   $AdresseEmail \longrightarrow UtilisateurID$
   En effet, à une adresse mail est associée un et un seul identificateur d'utilisateur.

   (d) *"Une adresse email est associée à un et un seul serveur de mails."*
   Cette contrainte s'exprime par la dépendance fonctionnelle :
   $AdresseEmail \longrightarrow ServeurMail$
   En effet, à une adresse mail est associée un et un seul serveur de mails.

2. **Identification des clés minimales de la relation R**

   La famille de dépendances fonctionnelles associées à R est :

   $$F = \{\ UtilisateurID \longrightarrow Nom, Prénom; \quad UtilisateurID, ServeurMail \longrightarrow Login, Passwd;$$
   $$AdresseEmail \longrightarrow UtilisateurID; \quad AdresseEmail \longrightarrow ServeurMail \quad \}$$

   L'attribut *AdresseEmail* ne peut être déduit d'aucun autre attribut, il doit donc appartenir à tous les clés minimales possibles de la relation. A partir de l'attribut *AdresseEmail* on peut déduire l'identificateur de l'Utilisateur est donc, par transitivité, le nom et le prénom de l'utilisateur : $AdresseEmail \longrightarrow UtilisateurID \longrightarrow Nom, Prénom$. A partir de ce même attribut, on peut en déduire aussi le nom du serveur de mail et donc avec l'identificateur d'utilisateur, le login et le mot de passe de l'utilisateur :
   $AdresseEmail \longrightarrow UtilisateurID, ServeurMail \longrightarrow Login, Passwd$
   D'où : $[AdresseEmail]^+ = \{\ AdresseEmail, UtilisateurID, Nom, Prénom, ServeurMail, Login, Passwd\ \} = R$

   La relation R a donc une seule clé minimale possible : *AdresseEmail*.

3. **Déduction de la forme normale du schéma de la relation R**

   Les deux dernières dépendances fonctionnelles sont de la forme *clé primaire* $\longrightarrow$ *autre attribut*, et vérifient donc les propriétés de la forme normale BCNF. En revanche, les deux premières dépendances fonctionnelles sont transitives puisqu'elles ne sont composées que d'attributs n'appartenant pas à une clé. Par conséquent, le schéma de la relation R est en deuxième forme normale.

Pour plus de détails sur les formes normales, vous pouvez vous référer aux pages 100 à 117 de l'ouvrage [1], au chapitre 15 de l'ouvrage [2] ou au chapitre 7 de l'ouvrage [3].

Bibliography

[1] H. Garcia-Molina, J.D. Ulmann et J. Widow, *Database Systems - The Complete Book*, Prentice Hall, 2002

[2] R. Ramakrishnan et J. Gehrke, *Database Management Systems*, Second Edition; McGraw-Hill, 2000, ISBN: 0-07-232206-3

[3] A. Silberschatz, H.F. Korth et S. Sudarshan, *Database System Concepts*, 4th Edition, McGraw-Hill, 2002, ISBN: 0-07-228363-7

[4] C. Soutou, *De UML à SQL - Conception de bases de données*, Eyrolles, 2002, ISBN: 2-212-11098-7