1 Cours : dépendances fonctionnelles et normalisations

1.1 Définitions

Une *dépendance fonctionnelle* est un énoncé du type

$$A_1, A_2, \ldots, A_k \to A_{k+1}, \ldots, A_n$$

où $A_1, A_2, \ldots, A_k, A_{k+1}, \ldots, A_n$ sont des attributs (colonnes) d'une base de données. Elle signifie que deux tuples ayant la même valeur sur $A_1, \ldots, A_k$ doivent avoir la même valeur sur chaque colonnes $A_{k+1}, \ldots, A_n$ (en français : $A_1, \ldots, A_k$ déterminent $A_{k+1}, \ldots, A_n$). On dit que les attributs $A_{k+1}, \ldots, A_n$ *dépendent fonctionnellement* de $A_1, A_2, \ldots, A_k$.

La notion de dépendance est transitive : si $A \to B$ et $B \to C$ alors $A \to C$. Un ensemble de dépendances fonctionnelles $\mathcal{F}$ est *minimal* si aucune dépendance ne peut être déduite des autres par transitivité.

La *clôture transitive* des attributs $A_1, \ldots, A_k$ pour un ensemble de dépendances fonctionnelles $\mathcal{F}$ est l'ensemble des attributs $B_1, \ldots, B_\ell$ qui dépendent fonctionnellement de $A_1, \ldots, A_k$. On la note $[A_1, \ldots, A_k]^{\mathcal{F}}_+$. Un ensemble d'attributs $A_1, \ldots, A_k$ est *une clef pour une relation* $R(B_1, \ldots, B_\ell)$ si ce sont des attributs de R et si sa clôture transitive contient $B_1, \ldots, B_\ell$. C'est une *clef primaire* si elle est minimale.

Un schéma est en :

- *Forme normale 1* si tout attribut est atomique (assuré par le modèle relationnel).
- *Forme normale 2* si un attribut ne fait pas partie d'une clef, il ne peut pas dépendre d'une partie stricte d'une clef.
- *Forme normale 3* Pour toute dépendance fonctionnelle, le membre de gauche contient une clef ou tout attribut du membre de droit appartient à une clef.

Un schéma et un ensemble de dépendances fonctionnelles peut se décomposer en une collection de schémas, dans le sens où chaque relation R peut se décomposer en $R_1, \ldots, R_k$ tels que $R_i = \pi_i(R)$ pour une certaine projection π_i.

On dit cette décomposition *sans perte d'information* si toute relation R du schéma d'origine peut être retrouvée à partir des relations $R_1, \ldots, R_k$: $R = \pi_1(R) \bowtie \ldots \bowtie \pi_k(R)$.

On dit que cette décomposition *respecte les dépendances fonctionnelles* si celles-ci sont toujours satisfaites par la nouvelle décomposition.

1.2 Exemple

Par exemple si $R(A, B, C, D)$ est une relation et $\mathcal{F} = \{ A \to B; B \to C; A \to C \}$ sont des dépendances fonctionnelles : $A \to C$ est redondante, on peut donc considérer seulement $\mathcal{F} = \{ A \to B; A \to C \}$. $[A]^{\mathcal{F}}_+ = \{ A, B, C \}$. A, D est une clef primaire de R. R est en forme normale 1. R n'est pas en FN 2 (car B ne fait pas partie d'une clef mais dépend d'une partie stricte d'une clef, de A). Ni en forme normale 3 : $A \to C$ le membre de gauche ne contient pas une clef, le membre de droit ne fait pas partie d'une clef.

$R(A, B, C, D)$ peut se décomposer en $R_1(A, D), R_2(A, B, C)$. A, D est une clef de R_1, A une clef de R_2. Ce nouveau schéma est en FN 1, en FN 2 et en FN 3. La décomposition se fait sans perte d'information et respecte les dépendances fonctionnelles. (si deux tuples de R_1 ont la même valeur sur A, alors ils ont la même valeur sur B et C. De même pour R_2).

2 TD

Nous voulons mettre en forme la base de données UFR dont le schéma est :

```
UFR(NoTD, Salle, Jour, Heure,
    NoEnseignant, Nom-Enseignant, Prenom-Enseignant,
    COD-MOD, Diplome, Matiere, NoEtudiant,
    Nom-Etudiant, Prenom-Etudiant, Adresse, Date-Inscription)
```

Sachant que :
- Un code module (COD−MOD) précise à la fois un diplôme et une matière.
- Les TD sont annuels et il y a un et un seul TD par semaine dans chaque module.
- Un TD est assuré par un seul enseignant.
- Un numéro de TD est relatif à un module.
- Un enseignant peut assurer plusieurs TD.
- Un étudiant peut être inscrit dans plusieurs modules, mais dans un seul TD par module.
- Date−Inscription est la date d'inscription d'un étudiant à un module.

Les dépendances fonctionnelles associées à ce schéma sont :

(a) NoEtudiant → Nom−Etudiant, Prenom−Etudiant, Adresse

(b) NoEnseignant→Nom−Enseignant, Prenom−Enseignant

(c) COD−MOD →Diplome,Matiere

(d) Diplome, matiere →COD−MOD

(e) Salle , Jour, Heure →NoTD, COD−MOD

(f) COD−MOD,NoTD→Salle , Jour, Heure, NoEnseignant

(g) COD−MOD,NoEtudiant→NoTD, Date−Inscription

(h) NoEnseignant, Jour, Heure→Salle

(i) NoEtudiant, Jour, Heure→Salle

(j) Salle , Jour, Heure → NoEnseignant

(k) NoEnseignant, COD−MOD, NoTD→Salle , Jour, Heure

1. Ce schéma a plusieurs anomalies. Donnez des exemples de redondances et de valeurs qui doivent être NULL à certains endroits.
2. L'ensemble des dépendances est long. Peut-être trop ?
   (a) Calculer les clôtures transitives de NoEnseignant et du triplet Diplome, NoEtudiant, matiere.
   (b) Les dépendances
   - Salle , Jour, Heure → NoEnseignant,
   - NoEnseignant, COD−MOD, NoTD→Salle , Jour, Heure
   - COD−MOD,NoTD→Salle

   sont-elles redondantes ? En déduire un ensemble minimal de dépendances fonctionnelles.
   (c) Donnez au moins trois clefs différentes de cette relation.

   On veut maintenant couper le schéma pour éviter ces anomalies. On veut que cette découpe vérifie les conditions suivantes :

   I Qu'elle se fasse sans pertes d'informations.

   II Qu'elle préserve les dépendances fonctionnelles.

   III Qu'elle n'ait plus besoin de NULL.

   IV Qu'elle minimise les redondances.

   V Et idéalement qu'elle respecte l'une des formes normales.
3. Donnez des clefs pour ces relations. Dites lesquelles des 5 conditions la décomposition suivante respecte. Dessinez les pattes de corbeaux associées.

```
TD(NoTD, COD-MOD, Jour, Heure, Salle, No-Enseignant,
   Nom-Enseignant, Prenom-Enseignant)

INSCRIPTION(NoEtudiant, Nom-Etudiant, Prenom-Etudiant, Adresse, COD-MOD,
            Diplome, Matiere, Date-Inscription, NoTD)
```

4. Donnez des clefs pour ces relations. Dites lesquelles des 5 conditions la décomposition suivante respecte. Dessinez les pattes de corbeaux associées.

```
ENSEIGNEMENT(NoTD, COD-MOD, Jour, Heure, Salle,
             No-Enseignant, Nom-Enseignant, Prenom-Enseignant)
ETUDIANT(NoEtudiant, Nom-Etudiant, Prenom-Etudiant, Adresse)
INSCRIPTION(NoEtudiant, COD-MOD, Date-Inscription, NoTD)
MODULE(COD-MOD, Diplome, Matiere)
```

5. Donnez des clefs pour ces relations. Dites lesquelles des 5 conditions la décomposition suivante respecte. Dessinez les pattes de corbeaux associées.

```
ENSEIGNEMENT_PLANNING(NoEnseignant, Nom-Enseignant,
                      Prenom-Enseignant, Jour, Heure, NoTD, COD-MOD)
ENSEIGNEMENT_SALLE(NoEnseignant, Jour, Heure, Salle)
ETUDIANT_INSCRIPTION(NoEtudiant, Nom-Etudiant, Prenom-Etudiant,
                     Adresse, NoTD, COD-MOD, Date-Inscription, Diplome, Matiere)
Etudiant_Planning(NoEtudiant, Jour, Heure, COD-MOD)
```

6. Proposez une décomposition qui satisfait les 5 conditions. Donnez les clefs et les pattes de corbeaux.

3 TP

3.1 Rappels

Dans ce TP, on va s'intéresser aux fonctions SQL. Nous donnons ici quelques rappels concernant la syntaxe des fonctions ainsi que certaines constructions dont vous aurez besoin pour la suite. Pour plus de détails, allez lire les transparents du cours de Stéphane du 31 mars 2015 (notamment le long exemple sur l'inscription d'un joueur à un tournoi dans la base ATP).

- Syntaxe générale pour la définition d'une fonction

```
CREATE OR REPLACE FUNCTION nom_de_la_fonction(parametre1, parametre2 ...)
RETURNS type AS
$$
BEGIN
corps de la fonction
END;
$$
LANGUAGE plpgsql;
```

- Tester si une requête renvoie quelque chose :

```
PERFORM requête;
IF FOUND THEN
  ́ce qu'il faut faire si ce n'est pas vide
ELSE
  ce qu'il faut faire si c'est vide
END IF;
```

- Attention, le `PERFORM` remplace le `SELECT`.
- Retourner une valeur :

```
RETURN valeur;
```

3.2 Exercice

On dispose d'une base de données pour gérer les présences aux différents événements et réunions des membres d'une association. Le schéma de la base de données est le suivant :

- `member(mid INT PRIMARY KEY, name VARCHAR(255), gender VARCHAR(1))`

- `location(lid INT PRIMARY KEY, address VARCHAR(255))`
- `event(eid INT PRIMARY KEY, lid INT, date DATE, hstart INT, hend INT)`
- `attend_to(mid INT, eid INT)`

qui contient les différents membres, les différents lieux où peuvent être organisés des événements, les différents évéments avec le lieu, la date, l'heure de début (entier entre 0 et 23) et l'heure de fin (entier entre 0 et 23, plus grand que hstart) ainsi qu'une table `attend_to` qui contient des couples `(m,e)` qui signifient que le membre d'identifiant `m` participera à l'événement `e`.

1. Implémentez cette base de données dans votre schéma avec les commandes suivantes (en remplaçant entid par votre identifiant ENT) :

```
CREATE TABLE entid.member
 (mid SERIAL PRIMARY KEY, name VARCHAR(255), gender VARCHAR(1));
CREATE TABLE entid.event
 (eid SERIAL PRIMARY KEY, lid INT, date DATE, hstart INT, hend INT);
CREATE TABLE entid.location
 (lid SERIAL PRIMARY KEY, address VARCHAR (255));
CREATE TABLE entid.attend_to
 (mid INT, eid INT);
```

2. Écrire des fonctions SQL qui réalisent les actions suivantes :
   (a) Une fonction `add_event(lid, date, hstart, hend)` qui ajoute un nouvel événement. Avant d'ajouter l'événement, la fonction doit vérifier que le lieu `lid` est bien disponible à cette date-là.
   (b) Une fonction `subscribe(mid,eid)` qui inscrit le membre `mid` à l'événement `eid`. La fonction doit vérifier que le membre n'a pas prévu de participer à un autre événement qui se déroule en même temps.
   (c) Une fonction `cancel(eid)` qui annule un événement. La fonction doit supprimer l'événement de la table `event` mais aussi nettoyer la table `attend_to`.
   (d) Une fonction `move_event(eid, lid, date, hstart, hend)` qui doit déplacer un événement. Votre fonction doit vérifier si le lieu est disponible aux nouveaux horaires et si les membres qui sont inscrits à cet événement sont tous disponibles aux nouveaux horaires.
3. Utilisez les triggers pour que la base de données aient les comportements suivants :
   (a) Lorsqu'on désire faire un `INSERT INTO event`, il faut que la fonction `add_event` soit appelée à la place.
   (b) Lorsqu'on désire faire un `INSERT INTO attend_to`, il faut que la fonction `subscribe` soit appelée à la place.
   (c) Lorsqu'on désire faire un `DELETE FROM event`, il faut que la fonction `cancel` soit appelée à la place.