université
PARIS-SACLAY
DIU EIL
Bloc 5

# Bases de données

## 1 Modélisation relationnelle

### Exercice 1

On rappelle la modélisation vue en cours :

```
CREATE TABLE Usager(uid INT PRIMARY KEY,
                    nom VARCHAR(255) NOT NULL,
                    prenom VARCHAR(255) NOT NULL,
                    datenaiss DATE NOT NULL,
                    adresse VARCHAR(255),
                    cp CHAR(5),
                    ville VARCHAR(50),
                    email VARCHAR(100) NOT NULL,
                    tel VARCHAR(20),
                    UNIQUE (nom, prenom, datenaiss, email),
                    CHECK (email LIKE '%@%'));

CREATE TABLE Abo(num INT PRIMARY KEY,
                 type VARCHAR(12),
                 datefin DATE,
                 zonedeb SMALLINT,
                 zonefin SMALLINT,
        CHECK (zonedeb >= 1 AND zonefin <= 5 AND zonedeb < zonefin),
        CHECK (type = 'mensuel' OR type = 'hebdomadaire' OR type = 'annuel'
               OR type = 'journalier'),
        CHECK (num >= 0));

CREATE TABLE Abo_Usager (num INT REFERENCES Abo,
                         uid INT REFERENCES Usager);
```

1. Proposer une modélisation relationnelle pour :
   — des gares (au minimum un nom de gare)
   — des passes navigo (ayant un numéro de passe, plusieurs passes pouvant être associé au même abonnement)
   — des bornes (ayant un numéro de série et se trouvant dans une gare)
   Cette modélisation devra permettre de réaliser l'action de validation : étant donné un passe, et une borne, la borne doit décider si elle autorise l'accès à la station. On poura procéder de la façon suivante :
   (a) Énumérer en français les caractéristiques de ces trois types d'entité, les relations qui le lient
   (b) Pour chacune des caractéristiques, donner le type de donnée abstrait (`Int`, `String`, ...) à utiliser et lister les contraintes en français
   (c) Donner enfin les commandes SQL permettant de créer les tables.
2. Pour chacune des nouvelles tables de la base donner, deux ordres SQL d'insertion violant des contraintes distinctes de la table. Vous pouvez supposer que des données sont déjà présentes, en explicitant lesquelles.

### Exercice 2

On considère des matches de foot entre des équipes de certaines villes, composées de joueurs. On souhaite pouvoir refléter les contraintes suivantes :

1. Proposer un ensemble d'ordre `CREATE` pour réaliser ces tables.
2. Décrire précisément en français ce qu'il faudrait vérifier sur les valeurs des tables pour que :
   — Un joueur ne puisse pas appartenir à deux équipes en même temps.
   — Une équipe ne puisse pas jouer deux matches le même jour
   (ces contraintes sont très difficilement exprimables uniquement avec des contraintes de table telles que `UNIQUE` et `CHECK`).

## 2 Requêtes

### Exercice 3

On se donne une base de données de films :

```
CREATE TABLE PEOPLE (pid INTEGER PRIMARY KEY,
                     firstname VARCHAR(30),
                     lastname VARCHAR(30));

CREATE TABLE MOVIE (mid INTEGER PRIMARY KEY,
                    title VARCHAR(90) NOT NULL,
                    year INTEGER NOT NULL,
                    runtime INTEGER NOT NULL,
                    rank INTEGER NOT NULL);

CREATE TABLE ROLE (mid INTEGER REFERENCES MOVIE,
                   pid INTEGER REFERENCES PEOPLE,
                   name VARCHAR(70));

CREATE TABLE DIRECTOR (mid INTEGER REFERENCES MOVIE,
                       pid INTEGER REFERENCES PEOPLE,
                       CONSTRAINT dir_const UNIQUE(mid,pid));
```

Pour cet exercice, il est possible d'utiliser la console dipsonible sur

`https://shell.nguyen.vg/sql/`

En utilisant les identifiants fournis. Dans cette console, il est possible de faire :

```
\i /opt/data/movies.sql
```

Pour créer les tables ci-dessus et les peupler de données.

Pour chacune des requêtes suivantes, donner un ordre SQL qui la calcule :

1. Renvoyer l'ensemble des films (*i.e.* le contenu de la table `movie`)
2. Renvoyer les films sortis en 2008
3. Renvoyer les films sortis entre 1990 et 2000 (au sens large)
4. Renvoyer les personnes dont le prénom est `Mike`
5. Renvoyer le nombre des personnes dont le prénom est `Mike`
6. Renvoyer le nombre de personnes dont le nom contient `ord`
7. Renvoyer les `pid` des personnes jouant dans le film dont le `mid` est `500`.
8. Renvoyer les prénoms et noms des personnes jouant dans le film dont le `mid` est `500`.
9. Renvoyer les prénoms et les noms des personnes jouant dans le film dont le titre est `'Batman Begins'`
10. Renvoyer la durée moyenne d'un film dans la base.

11. Renvoyer les films dont l'année de sortie est bissextile (divisible par 4, mais pas par 100 ou alors par 400).
12. Renvoyer les `mid`, les titres et les durées des films dont la durée est inférieure à la durée du film dont le titre est `'Star Wars'`.
13. (difficile) Renvoyer les titres des films ayant au moins deux réalisateurs
14. Renvoyer les titres des films ou un des réalisateurs joue un role (en utilisant une jointure)
15. Comme la précédente en utilisant une requête imbriquée.
16. Les années de la table `movie`, sans doublon.
17. (difficile) les acteurs qui jouent dans le même film ainsi que le titre du film. On veut des couples de noms et prénoms de joueurs $n_1, p_1, n_2, p_2, t$ tels que les deux acteurs étaient dans la même film de titre $t$. De plus si $n_1, p_1, n_2, p_2, t$ est dans le résultat, on ne veut pas que $n_2, p_2, n_1, p_1, t$ y soit aussi.

### Exercice 4

On reprend les tables créées à l'exercice 2. Pour chacune des requêtes suivantes, donner un ordre SQL qui la calcule :

1. Les villes des équipes dont 'Hugo Lloris' a été membre.
2. Les villes des équipes ayant gagné un match entre le premier janvier et le 31 décembre 2016.

## 3 Mises à jour

### Exercice 5

Pour chacune des mises à jour suivantes sur la base de données des films, donner les ordres SQL permettant de les réaliser ces mises à jour. Attention, il faut parfois plusieurs ordres SQL successifs pour la même mise à jour.

Il est possible d'utiliser la console SQL de l'exercice 3 pour tester les réponses. Dans ce cas, on peut à tout moment revenir à un état initial en rechargeant le script `movies.sql` comme indiqué dans l'exercice 2.

1. Ajouter 100 à tous les `rang` (de film) supérieurs à `100`.
2. Supprimer tous les rôles dont le nom est `Jim`
3. Pour le rôle dont le nom est `'Neo'` du film `'The Matrix'`, modifier le nom du rôle en `'The One'`
4. Supprimer le film dont le titre est `'Star Wars'`
5. Changer le `mid` du film `'It'` de `1170` à `2000`.