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.

Réponse:

1. (a) On commence par décrire les entités

   Gare : la gare doit avoir un nom, et une zone. On ajoute un `gid` identifiant unique. On pourrait ajouter une adresse, des coordonnées GPS,....

   Passe : un passe doit avoir un numéro.

   Borne : une borne doit avoir un numéro.

   Gare_Borne : représente les bornes se trouvant dans une gare

   Passe_Abo : les passes associés à un abonnement.

   (b)
   - Le nom de la gare est de type `String`, l'identifiant et les zones de type `Int`. L'identifiant est une clé primaire.
   - Le numéro de passe est de type `String` et est une clé primaire.
   - Le numéro de borne est de type `String` et est une clé primaire.
   - `Gare_Borne` associe des identifiants de gares et des identifiants de bornes (clés étrangères)
   - `Passe_Abo` associe des identifiants de passes et d'abonnements (clés étrangères)
   - Les numéros de zones sont compris entre 1 et 5
   - Un passe ne peut pas être associé à deux abonnements.
   - Une borne ne peut se trouver que dans une gare.

2. Cela donne en SQL

```
CREATE TABLE Gare (gid INT PRIMARY KEY,
                   nom VARCHAR(255) NOT NULL,
                   zone SMALLINT,
                   CHECK (zone >= 1 AND zone <= 5));

CREATE TABLE Passe (pid VARCHAR(16) PRIMARY KEY);

CREATE TABLE Borne (bid VARCHAR(16) PRIMARY KEY);

CREATE TABLE Gare_Borne(gid INT REFERENCES Gare,
                        bid VARCHAR(16) REFERENCES Borne UNIQUE);

CREATE TABLE Passe_Abo(pid VARCHAR(16) REFERENCES Passe UNIQUE,
                       num INT REFERENCES Abo);
```

```
INSERT INTO Gare (1, 'Le␣Guichet', 9); -- contrainte CHECK

INSERT INTO Gare (1, 'Le␣Guichet', 5); -- ok
INSERT INTO Gare (1, 'Le␣Guichet', 5); -- contrainte de clé primaire


INSERT INTO Passe ('aaaaaaaaaaaaaaaaaaaaa'); -- contrainte domaine

INSERT INTO Passe ('abcd'); -- ok
INSERT INTO Passe ('abcd'); -- contrainte de clé primaire


INSERT INTO Borne ('aaaaaaaaaaaaaaaaaaaaa'); -- contrainte domaine

INSERT INTO Borne ('abcd'); -- ok
INSERT INTO Borne ('abcd'); -- contrainte de clé primaire
```

```
INSERT INTO Gare_Borne (1, 'abcd'); -- ok
INSERT INTO Gare (2, 'Orsay-Ville', 5); -- ok
INSERT INTO Gare_Borne(2, 'abcd') ; -- contrainte UNIQUE
INSERT INTO Gare_Borne(3, 'abcd'); --contrainte clé étrangère

-- similaire pour Pass_Abo
```

Exercice 2

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

3. Proposer un ensemble d'ordre `CREATE` pour réaliser ces tables.

Réponse:

```
CREATE TABLE Joueur (jid INTEGER PRIMARY KEY,
                     nom VARCHAR(200),
                     prenom VARCHAR(200));
CREATE TABLE Equipe (eid INTEGER PRIMARY KEY,
                     ville VARCHAR (50),
                     cp VARCHAR (5), UNIQUE(ville, cp));
CREATE TABLE Joueur_Equipe(jid INTEGER REFERENCES Joueur,
                           eid INETEGER REFERENCES Equipe,
                           d_debut DATE,
                           d_fin DATE,
                           CHECK (d_fin > d_debut));
CREATE TABLE Match (eid1 INTEGER REFERENCES Equipe(eid),
                    eid2 INTEGER REFERENCES Equipe(eid),
                    score1 INTEGER,
                    score2 INTEGER,
                    date DATE,
                    CHECK(score1 >= 0 AND score2 >= 0));
```

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`).

Réponse:

— Dans la table `Joueur_Equipe` on veut que si deux lignes (j, e_1, d_1, f_1) et (j, e_2, d_2, f_2) soient existent (pour le même j) alors les intervalles $[d_1, f_1]$ et $[d_2, f_2]$ sont disjoints).

— On doit vérifier que si (e_1, e_2, d) appartient à `Match` alors il n'existe pas e_3 tel que : (e_1, e_3, d) appartient à `Match` ou bien (e_3, e_1, d) appartient à `Match`.

2 Requêtes

Exercice 3

On se donne une base de données de films :

```sql
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.

Réponse:

```sql
-- 1
SELECT * FROM movie;

-- 2
SELECT * FROM movie WHERE year = 2008;

-- 3
SELECT * FROM movies WHERE year >= 1990 AND year <= 2000;

-- 4
SELECT * FROM people WHERE firstname = 'Mike';

-- 5

SELECT COUNT(*) FROM people WHERE firstname = 'Mike';

-- 6

SELECT COUNT(*) FROM people WHERE lastname LIKE '%ord%';

-- 7

SELECT pid FROM role WHERE mid = 500;

-- 8

SELECT p.firstname, p.lastname FROM people p, role r
                               WHERE r.mid = 500 AND
                               p.pid = m.pid;
            -- ou --
SELECT p.firstname, p.lastname FROM people p
                               JOIN role r ON p.pid = r.pid
                               WHERE r.mid = 500;


-- 9
SELECT p.firstname, p.lastname FROM people p
                               JOIN role r ON p.pid = r.pid
                               JOIN movie m ON r.mid = m.mid
                               WHERE m.title = 'Batman␣Begins';

-- 10
SELECT AVG(runtime) FROM movie;

-- 11
SELECT * FROM movie WHERE (MOD(year,4) = 0 AND MOD (year, 100) <> 0)
                          OR MOD(year, 400) = 0;

-- 12
SELECT mid, title, runtime FROM movie
WHERE runtime < (SELECT runtime FROM movie
                 WHERE title = 'Star␣Wars');
```

```sql
-- 13
-- RENVOYER LE NOMBRE DE RÉALISATEUR D'UN FILM
-- DONT LE MID '123' EST DONNÉ
-- SELECT COUNT(*) FROM director d
--                 WHERE d.mid = '123';

-- On en déduit :

SELECT * FROM movie m
WHERE (SELECT COUNT (*) FROM director d
                        WHERE d.mid = m.mid) >= 2;


-- 14
SELECT m.title FROM movie m
               JOIN role r ON r.mid = m.mid
               JOIN director d ON d.mid = r.mid
               WHERE r.pid = d.pid;


-- 15
SELECT m.title FROM movie m
               JOIN director d ON d.mid = m.mid
               WHERE d.pid IN (SELECT r.pid FROM role r
                                            WHERE r.mid = m.mid);


-- 16
SELECT DISTINCT year FROM movie;

-- 17
SELECT p1.lastname, p1.firstname, p2.lastname, p2.firstname, m.title
FROM role r1
JOIN role r2 ON r1.mid = r2.mid
JOIN people p1 ON r1.pid = p1.pid
JOIN people p2 ON r2.pid = p2.pid
JOIN movie m ON r1.mid = m.mid
WHERE r1.pid < r2.pid;
```

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.

Réponse:

```sql
-- 1
SELECT e.ville FROM Equipe e
               JOIN Equipe_Joueur ej ON ej.eid = e.eid
               JOIN Joueur j ON j.jid = ej.jid
               WHERE j.nom = 'Lloris' AND j.prenom = 'Hugo';
```

```sql
-- 2
SELECT e.ville FROM Equipe e, Match m
WHERE ((e.eid = m.eid1 AND m.score1 > m.score2)
      OR (e.eid = m.eid2 AND m.score1 < m.score2))
AND m.date >= '2016-01-01' ANDm.date <= '2016-12-31';
```

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`.

Réponse:

```sql
-- 1
UPDATE movie SET rank = rank + 100 WHERE rank >= 100;

-- 2
DELETE FROM role WHERE name = 'Jim';

-- 3
UPDATE role SET name = 'The One'
WHERE name = 'Neo'
  AND mid = (SELECT mid FROM movie WHERE title = 'The Matrix');

-- 4
-- d'abord supprimer les roles et réalisateurs
-- On stocke le mid de starwards dans une table temporaire.

SELECT mid INTO sw_mid FROM movie WHERE title = 'Star Wars';
DELETE FROM role WHERE mid = (SELECT * FROM sw_mid);
DELETE FROM director WHERE mid = (SELECT * FROM sw_mid);
DELETE FROM movie WHERE mid = (SELECT * FROM sw_mid);
DROP TABLE sw_mid;


-- 5
SELECT * INTO it_movie FROM movie WHERE title = 'It';
UPDATE it_movie SET mid = 2000;
INSERT INTO movie SELECT * FROM it_movie;
```

```
UPDATE role SET mid = 2000 WHERE mid = 1170;
UPDATE director SET mid = 2000 WHERE mid = 1170;
DELETE FROM movie WHERE mid = 1170;
DROP TABLE it_movie;
```