INFO0009 - Bases de données

Répétition 6 : Le langage SQL

Maxime Meurisse (slides de Remy Vandaele)
m.meurisse@student.uliege.be

2021

Bref rappel

Le SQL est un **langage** de gestion de BD relationnelles :

- Création et modification de relations (Tables en SQL)
- Opérations d'algèbre relationnel (Requêtes SQL)

Il permet de **communiquer** avec un **SGBD** :

- Interprète et exécute la requête SQL
- Maintient la BD
- MySQL, SQLite, Access,...

Exercice 1

1. Personne(n_national, nom, prenom, sexe, rue, numero, #code_postal, #code_localite)
2. Commune(code_postal,nom_commune)
3. Localite(#code_postal,code_localite,nom_localite)
4. Mariage(code_mariage, #n_national_A, #n_national_B, date_mariage, date_divorce, #code_postal)
5. Enfant(#n_national_enfant, #n_national_A, #n_national_B, date_naissance)

Exercice 1

1. Création de tables

Format général :

```
CREATE TABLE IF NOT EXISTS <NOM_RELATION>(
    <nom_att1> <type_att1> <option_att1>,
    ...
    <nom_attN> <type_attN> <option_attN>,
    PRIMARY KEY(nom_attI),
    FOREIGN KEY(nom_attJ) REFERENCES <REL.>(<nom_att>),
    ...
    FOREIGN KEY(nom_attK) REFERENCES <REL.>(<nom_att>)
)ENGINE=InnoDB;
```

Type : INT,BIGINT,VARCHAR,DATE,...
Options : NOT NULL, PRIMARY KEY, AUTO_INCREMENT,...

Exercice 1

1. Création de tables

NOT NULL : L'attribut doit toujours avoir une valeur quand un tuple est ajouté.
AUTO_INCREMENT : La valeur de l'attribut s'incrémente quand un tuple est ajouté (utile pour des clés primaires).
PRIMARY KEY : Clé primaire.
FOREIGN KEY X REFERENCES R(Y) : En ajoutant un tuple dans la table, la valeur de l'attribut X DOIT être présente dans un tuple de la table R, dans la colonne (attribut) Y.

Exercice 1

1. Création de tables

Créer les tables suivantes :

1. Personne(n_national, nom, prenom, sexe, rue, numero, #code_postal, #code_localite)
2. Commune(code_postal,nom_commune)
3. Localite(#code_postal,code_localite,nom_localite)
4. Mariage(code_mariage, #n_national_A, #n_national_B, date_mariage, date_divorce, #code_postal)
5. Enfant(#n_national_enfant, #n_national_A, #n_national_B, date_naissance)

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
```

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
    n_national BIGINT PRIMARY KEY,
```

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
    n_national BIGINT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
```

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
    n_national BIGINT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
```

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
    n_national BIGINT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    sexe VARCHAR(1) NOT NULL,
```

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
    n_national BIGINT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    sexe VARCHAR(1) NOT NULL,
    rue VARCHAR(50) NOT NULL,
```

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
    n_national BIGINT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    sexe VARCHAR(1) NOT NULL,
    rue VARCHAR(50) NOT NULL,
    numero VARCHAR(5) NOT NULL,
```

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
    n_national BIGINT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    sexe VARCHAR(1) NOT NULL,
    rue VARCHAR(50) NOT NULL,
    numero VARCHAR(5) NOT NULL,
    code_postal INT NOT NULL,
```

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
    n_national BIGINT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    sexe VARCHAR(1) NOT NULL,
    rue VARCHAR(50) NOT NULL,
    numero VARCHAR(5) NOT NULL,
    code_postal INT NOT NULL,
    code_localite INT NOT NULL
```

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
    n_national BIGINT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    sexe VARCHAR(1) NOT NULL,
    rue VARCHAR(50) NOT NULL,
    numero VARCHAR(5) NOT NULL,
    code_postal INT NOT NULL,
    code_localite INT NOT NULL,
    FOREIGN KEY (code_postal,code_localite)
        REFERENCES Localite(code_postal,code_localite)
```

Exercice 1

1. Création de tables

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)

```
CREATE TABLE IF NOT EXISTS Personne(
    n_national BIGINT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    sexe VARCHAR(1) NOT NULL,
    rue VARCHAR(50) NOT NULL,
    numero VARCHAR(5) NOT NULL,
    code_postal INT NOT NULL,
    code_localite INT NOT NULL,
    FOREIGN KEY (code_postal,code_localite)
        REFERENCES Localite(code_postal,code_localite)
)engine=InnoDB;
```

Exercice 1

1. Création de tables

Créer les tables suivantes :

1. Personne(n_national, nom, prenom, sexe, rue, numero, #code_postal, #code_localite)
2. Commune(code_postal,nom_commune)
3. Localite(#code_postal,code_localite,nom_localite)
4. Mariage(code_mariage, #n_national_A, #n_national_B, date_mariage, date_divorce, #code_postal)
5. Enfant(#n_national_enfant, #n_national_A, #n_national_B, date_naissance)

```
CREATE TABLE IF NOT EXISTS <NOM_REL.>(
    <nom_att1> <type_att1> <option_att1>,
    ...
    <nom_attN> <type_attN> <option_attN>,
    PRIMARY KEY(nom_attI),
    FOREIGN KEY(nom_attJ) REFERENCES <TABLE>(<nom_rel>),
    ...
    FOREIGN KEY(nom_attK) REFERENCES <TABLE>(<nom_rel>)
)ENGINE=InnoDB;
```

Commune(code_postal,nom_commune)

```
CREATE TABLE IF NOT EXISTS Commune(
    code_postal INT PRIMARY KEY,
    nom_commune VARCHAR(50) NOT NULL
)ENGINE=InnoDB;
```

Localite(code_postal,code_localite,nom_localite)

```
CREATE TABLE IF NOT EXISTS Localite(
  code_postal INT NOT NULL,
  code_localite INT NOT NULL,
  nom_localite VARCHAR(50) NOT NULL,
  PRIMARY KEY (code_postal,code_localite),
  FOREIGN KEY code_postal REFERENCES Commune(code_postal)
)ENGINE=InnoDB;
```

Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)

```
CREATE TABLE IF NOT EXISTS Mariage(
    code_mariage INT PRIMARY KEY,
    n_national_A BIGINT NOT NULL,
    n_national_B BIGINT NOT NULL,
    date_mariage DATE NOT NULL,
    date_divorce DATE,
    code_postal INT NOT NULL,
    FOREIGN KEY code_postal
        REFERENCES Commune(code_postal),
    FOREIGN KEY n_national_A
        REFERENCES Personne(n_national),
    FOREIGN KEY n_national_B
        REFERENCES Personne(n_national),
    CONSTRAINT CHECK_DATES
        CHECK (date_divorce >= date_mariage)
)ENGINE=InnoDB;
```

Enfant(n_national_enfant, n_national_A, n_national_B, date_naissance)

```
CREATE TABLE IF NOT EXISTS Enfant(
    n_national_enfant BIGINT NOT NULL,
    n_national_A BIGINT,
    n_national_B BIGINT NOT NULL,
    date_naissance DATE NOT NULL,
    PRIMARY KEY n_national_enfant,
    FOREIGN KEY n_national_enfant
        REFERENCES Personne(n_national),
    FOREIGN KEY n_national_A
        REFERENCES Personne(n_national),
    FOREIGN KEY n_national_B
        REFERENCES Personne(n_national)
)ENGINE=InnoDB;
```

Exercice 1

2. Requêtes

Exercice 1

2. Requêtes

Format de base d'une requête :

```
SELECT <att_i>,...,<att_k> | *
FROM <nom_table>
```

- Sélectionne tous les tuples de la table
- Les tuples sont composés des attributs (noms de colonnes)
  - Tous si *
- Possible d'utiliser une fonction dans le select (DISTINCT, COUNT, SUM, AVG,...)
- Le résultat d'une requête sera une table

Exercice 1

2. Requêtes

Table ETUDIANT :

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16
Durant	Francis	17
Durant	Gustave	17
Leclercq	Alain	14

```
SELECT Nom,Prénom
FROM ETUDIANT
```

Nom	Prénom
Dupont	Georges
Dupont	Henri
Durant	Francis
Durant	Gustave
Leclercq	Alain

Exercice 1

2. Requêtes

Table ETUDIANT :

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16
Durant	Francis	17
Durant	Gustave	17
Leclercq	Alain	14

```
SELECT DISTINCT Nom
FROM ETUDIANT
```

Nom
Dupont
Durant
Leclercq

Exercice 1

2. Requêtes

Exemple :

Table ETUDIANT :

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16
Durant	Francis	17
Durant	Gustave	17
Leclercq	Alain	14

```
SELECT AVG(Age)
FROM ETUDIANT
```

AVG(Age)
16.2

Exercice 1

2. Requêtes

```
SELECT <att_i> AS nouveau_nom_colonne
FROM <nom_table>
```

- AS permet de renommer une colonne
- Utile pour les jointures et les unions

Exercice 1

2. Requêtes

Exemple :

Table ETUDIANT :

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16
Durant	Francis	17
Durant	Gustave	17
Leclercq	Alain	14

```
SELECT AVG(Age) AS MOYAGE
FROM ETUDIANT
```

MOYAGE
16.2

Exercice 1

2. Requêtes

WHERE (optionnel) :

```
SELECT *
FROM <nom_table>
WHERE CSTR(column)
```

WHERE impose des contraintes sur les tuples en fonction des valeurs des attributs

- $>, <, =$,(NOT) IN, (NOT) EXISTS

Exercice 1

2. Requêtes

Table ETUDIANT :

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16
Durant	Francis	17
Durant	Gustave	17
Leclercq	Alain	14

```
SELECT *
FROM ETUDIANT
WHERE Nom="Dupont"
```

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16

Exercice 1

2. Requêtes

TABLE ETUDIANT :

ID	Nom	Prénom
1	Leclercq	Alain
2	Durant	Georges
3	Vermeulen	Eddy
4	Delbecq	Nadine

TABLE INTERRO :

ID	Cours	Points
1	Math	16
2	Francais	12
3	Geographie	17
4	Francais	14
3	Math	11

```
SELECT Nom,Prénom
FROM ETUDIANT
WHERE ID IN
    SELECT ID FROM INTERRO WHERE Points>15
```

Nom	Prénom
Leclercq	Alain
Vermeulen	Eddy

Exercice 1

2. Requêtes

TABLE ETUDIANT :

ID	Nom	Prénom
1	Leclercq	Alain
2	Durant	Georges
3	Vermeulen	Eddy
4	Delbecq	Nadine

TABLE INTERRO :

ID	Cours	Points
1	Math	16
2	Francais	12
3	Geographie	17
4	Francais	14
3	Math	11

```
SELECT Nom,Prénom
FROM ETUDIANT e
WHERE NOT EXISTS
    SELECT * FROM INTERRO WHERE (Points<15 AND e.ID=ID)
```

Nom	Prénom
Leclercq	Alain

Exercice 1

2. Requêtes

```
SELECT *
FROM <nom_table_1> NATURAL JOIN <nom_table_2>
```

NATURAL JOIN fait le produit cartésien des entrées de deux tables en se limitant aux tuples dont les entrées ont la même valeur pour les attributs de même nom.

Exercice 1

2. Requêtes

TABLE ETUDIANT :

ID	Nom	Prénom
1	Leclercq	Alain
2	Durant	Georges

TABLE INTERROS :

ID	Cours	Points
1	Math	17
1	Francais	10
2	Geographie	14
2	Francais	12

```
SELECT *
FROM ETUDIANT NATURAL JOIN INTERROS
```

ID	Nom	Prénom	Cours	Points
1	Leclercq	Alain	Math	17
1	Leclercq	Alain	Francais	12
2	Durant	Georges	Geographie	14
2	Durant	Georges	Francais	12

Exercice 1

2. Requêtes

```
SELECT *
FROM <NOM_TABLE>
GROUP BY <colonne>
```

GROUP BY regroupe les résultats par un ou plusieurs attributs de même valeur

Requêtes

TABLE ETUDIANT :

Nom	Prénom	Age	Points
Dupont	Georges	17	12
Dupont	Henri	16	13
Durant	Francis	17	15
Durant	Gustave	17	14
Leclercq	Alain	14	12

```
SELECT Nom, SUM(Age), AVG(Points)
FROM ETUDIANT
GROUP BY Nom
```

Nom	SUM(Age)	AVG(Points)
Dupont	33	12.5
Durant	34	14.5
Leclercq	14	12

Exercice 1

2. Requêtes

```
SELECT *
FROM <NOM_TABLE>
GROUP BY <colonne>
HAVING contrainte_groupe
```

HAVING permet de restreindre les groupes en leur imposant des contraintes

Requêtes

TABLE ETUDIANT :

Nom	Prénom	Age	Points
Dupont	Georges	17	12
Dupont	Henri	16	13
Durant	Francis	17	15
Durant	Gustave	17	14
Leclercq	Alain	14	12

```
SELECT Nom, SUM(Age), AVG(Points)
FROM ETUDIANT
GROUP BY Nom
HAVING AVG(Points)<13
```

Nom	SUM(Age)	AVG(Points)
Dupont	33	12.5
Leclercq	14	12

Exercice 1

2. Requêtes

```
TABLE_1 UNION TABLE_2
```

UNION permet de regrouper deux tables en une seule (si les colonnes sont de même nom)

Exercice 1

2. Requêtes

TABLE Etudiant :

Nom	**Prénom**	Grade
Dupont	Henri	PGD
Vermeiren	Nadine	S

TABLE Professeur :

Nom	**Prénom**	**Cours**
Delobelle	Peter	Informatique-1
Deroy	Marc	Analyse numérique

```
(SELECT Nom,Prénom
FROM Etudiant)
UNION
(SELECT Nom,Prénom
FROM Professeur)
```

Nom	**Prénom**
Dupont	Henri
Vermeiren	Nadine
Delobelle	Peter
Deroy	Marc

Exercice 1

2. Requêtes

- Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)
- Commune(code_postal,nom_commune)
- Localite(code_postal,code_localite,nom_localite)
- Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)
- Enfant(n_national_enfant, n_national_A, n_national_B, date_naissance)

(a) Nom et prénom des personnes s'étant mariées à Liège

(b) Nom et prénom des personnes sans enfants

(c) Nom et prénom des personnes nées de personne B inconnue

(d) Nombre de localités par communes

(e) Nom et prénom des enfants nés après le mariage de leurs parents

(f) Pourcentage de divorces par communes

(a) Nom et prénom des personnes s'étant mariées à Liège (I)

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal,code_localite)
Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)

1. Sélection des id des personnes (A) et des personnes (B) s'étant mariées à Liège.

```
(SELECT n_national_A AS nn
FROM Mariage
WHERE code_postal=4000)

UNION

(SELECT n_national_B AS nn
FROM Mariage
WHERE code_postal=4000)
```

(a) Nom et prénom des personnes s'étant mariées à Liège (II)

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal,code_localite)
Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)

2. Sélection des noms et prénoms des personnes dont l'identifiant a été retrouvé en (1).

```
(SELECT n_national_A AS nn
FROM Mariage
WHERE code_postal=4000)

UNION

(SELECT n_national_B AS nn
FROM Mariage
WHERE code_postal=4000)
```

(a) Nom et prénom des personnes s'étant mariées à Liège (II)

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal,code_localite)
Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)

2. Sélection des noms et prénoms des personnes dont l'identifiant a été retrouvé en (1).

```
SELECT nom, prenom
FROM Personne
WHERE n_national IN
    (
     (SELECT n_national_A AS nn
     FROM Mariage
     WHERE code_postal=4000)

     UNION

     (SELECT n_national_B AS nn
     FROM Mariage
     WHERE code_postal=4000)
    )
```

- Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)
- Commune(code_postal,nom_commune)
- Localite(code_postal,code_localite,nom_localite)
- Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)
- Enfant(n_national_enfant, n_national_A, n_national_B, date_naissance)

(a) Nom et prénom des personnes s'étant mariées à Liège

(b) Nom et prénom des personnes sans enfants

(c) Nom et prénom des personnes nées de personne B inconnue

(d) Nombre de localités par communes

(e) Nom et prénom des enfants nés après le mariage de leurs parents

(f) Pourcentage de divorces par communes

(b) Nom et prénom des personnes sans enfants

Personne(<u>n_national</u>, nom, prenom, sexe, rue, numero, code_postal, code_localite)
Enfant(<u>n_national_enfant</u>, n_national_A, n_national_B, date_naissance)

```
(SELECT n_national_B AS nn
FROM Enfant)

UNION

(SELECT n_national_A AS nn
FROM Enfant)
```

(b) Nom et prénom des personnes sans enfants

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)
Enfant(n_national_enfant, n_national_A, n_national_B, date_naissance)

```
SELECT nom, prenom
FROM Personne
WHERE n_national NOT IN
    (
        (SELECT n_national_B AS nn
        FROM Enfant)

        UNION

        (SELECT n_national_A AS nn
        FROM Enfant)
    )
```

(c) Nom et prénom des personnes nées de personne B inconnue

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)
Enfant(n_national_enfant, n_national_A, n_national_B, date_naissance)

```
SELECT n_national_enfant
FROM Enfant
WHERE n_national_B IS NULL
```

(c) Nom et prénom des personnes nées de personne B inconnue

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)
Enfant(n_national_enfant, n_national_A, n_national_B, date_naissance)

```
SELECT nom, prenom
FROM Personne
WHERE n_national IN
    (SELECT n_national_enfant
    FROM Enfant
    WHERE n_national_B IS NULL)
```

(d) Nombre de localités par commune

Commune(code_postal,nom_commune)
Localite(code_postal,code_localite,nom_localite)

```
SELECT nom_commune, COUNT(code_localite) AS n_localites
FROM Commune NATURAL JOIN Localite
GROUP BY code_postal
```

(e) Nom et prénom des enfants nés après le mariage de leurs parents (I)

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)
Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)
Enfant(n_national_enfant, n_national_A, n_national_B, date_naissance)

1. Sélectionner le numéro national des enfants nés après le mariage de leurs parents

```
SELECT n_national_enfant
FROM Enfant NATURAL JOIN
    (SELECT n_national_A , n_national_B, date_mariage FROM Mariage)
WHERE date_naissance>date_mariage
```

(e) Nom et prénom des enfants nés après le mariage de leurs parents (II)

Personne(n_national, nom, prenom, sexe, rue, numero, code_postal, code_localite)
Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)
Enfant(n_national_enfant, n_national_A, n_national_B, date_naissance)

2. Trouver les noms et prénoms des personnes dont on a obtenu le numéro national

```
SELECT nom, prenom
FROM Personne
WHERE n_national IN
    (SELECT n_national_enfant
    FROM Enfant NATURAL JOIN
        (SELECT n_national_A, n_national_B, date_mariage FROM Mariage)
    WHERE date_naissance>date_mariage)
```

(f) Pourcentage de divorces par commune (I)

Commune(code_postal,nom_commune)
Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)

1. Nombre de mariages par code postal

```
SELECT code_postal, COUNT(*) AS nmariages
FROM Mariage
GROUP BY code_postal
```

2. Nombre de divorces par code postal

```
SELECT code_postal, COUNT(*) AS ndivorces
FROM Mariage
WHERE date_divorce IS NOT NULL
GROUP BY code_postal
```

(f) Pourcentage de divorces par commune (II)

Commune(code_postal,nom_commune)
Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)

3. Nombre de mariages et de divorces par code postal

```
SELECT code_postal, nmariages, ndivorces
FROM
    (SELECT code_postal, COUNT(*) AS nmariages
    FROM Mariage
    GROUP BY code_postal)

    NATURAL JOIN

    (SELECT code_postal, COUNT(*) AS ndivorces
    FROM Mariage
    WHERE date_divorce IS NOT NULL
    GROUP BY code_postal)
```

(f) Pourcentage de divorces par commune (III)

Commune(code_postal,nom_commune)
Mariage(code_mariage, n_national_A, n_national_B, date_mariage, date_divorce, code_postal)

4. NATURAL JOIN pour les noms de commune, nmariages$>$ 0

```
SELECT nom_commune, ndivorces/nmariages
FROM
    SELECT code_postal, nmariages, ndivorces
    FROM
        (
            (SELECT code_postal, COUNT(code_mariage) AS nmariages
            FROM Mariage
            GROUP BY code_postal)

            NATURAL JOIN

            (SELECT code_postal, COUNT(code_mariage) AS ndivorces
            FROM Mariage
            WHERE date_divorce IS NOT NULL
            GROUP BY code_postal)
        )

        NATURAL JOIN

        Commune
```

Exercice 2

1. Création de tables

Créer les tables suivantes :

1. Auteur(code_auteur, nom, prenom)
2. Livre(n_livre, ISBN, titre, editeur, date_edition)
3. Ecrit(#code_auteur, #n_livre)
4. Exemplaire(#n_livre, n_exemplaire, etat, date_acq)
5. Emprunteur(n_inscr, nom, prenom, adresse)
6. Emprunt(#n_livre, #n_exemplaire, #n_inscr, date_empr)

```
CREATE TABLE IF NOT EXISTS <NOM_REL.>(
    <nom_att1> <type_att1> <option_att1>,
    ...
    <nom_attN> <type_attN> <option_attN>,
    PRIMARY KEY(nom_attI),
    FOREIGN KEY(nom_attJ) REFERENCES <TABLE>(<nom_rel>),
    ...
    FOREIGN KEY(nom_attK) REFERENCES <TABLE>(<nom_rel>)
)ENGINE=InnoDB;
```

Auteur(code_auteur, nom, prenom)

```
CREATE TABLE IF NOT EXISTS Auteur(
    code_auteur INT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL
)ENGINE=InnoDB;
```

Livre(n_livre, ISBN, titre, date_edition)

```
CREATE TABLE IF NOT EXISTS Livre(
    n_livre INT PRIMARY KEY,
    ISBN VARCHAR(17) UNIQUE,
    editeur VARCHAR(50) NOT NULL,
    titre VARCHAR(100) NOT NULL,
    date_edition DATE NOT NULL
)ENGINE=InnoDB;
```

Ecrit(code_auteur, n_livre)

```
CREATE TABLE IF NOT EXISTS Ecrit(
    code_auteur INT,
    n_livre INT,
    PRIMARY KEY (code_auteur,n_livre),
    FOREIGN KEY code_auteur REFERENCES Auteur(code_auteur),
    FOREIGN KEY n_livre REFERENCES Livre(n_livre)
)ENGINE=InnoDB;
```

Exemplaire(n_livre, n_exemplaire, etat, date_acq)

```
CREATE TABLE IF NOT EXISTS Exemplaire(
    n_livre INT PRIMARY KEY,
    n_exemplaire INT PRIMARY KEY,
    etat VARCHAR(50) NOT NULL,
    date_acq DATE NOT NULL,
    FOREIGN KEY n_livre REFERENCES Livre(n_livre)
)ENGINE=InnoDB;
```

Emprunteur(n_inscr, nom, prenom, adresse)

```
CREATE TABLE IF NOT EXISTS Emprunteur(
    n_inscr INT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    adresse VARCHAR(200) NOT NULL,
)ENGINE=InnoDB;
```

Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)

```
CREATE TABLE IF NOT EXISTS Emprunt(
  n_livre INT PRIMARY KEY,
  n_exemplaire INT PRIMARY KEY,
  n_inscr INT NOT NULL,
  date_empr DATE NOT NULL,
  FOREIGN KEY (n_livre,n_exemplaire) REFERENCES
      Exemplaire(n_livre,n_exemplaire) ,
  FOREIGN KEY n_inscr REFERENCES Emprunteur(n_inscr)
)ENGINE=InnoDB;
```

Exercice 2

2. Requêtes

- Auteur(code_auteur, nom, prenom)
- Livre(n_livre, ISBN, titre, editeur, date_edition)
- Ecrit(code_auteur, n_livre)
- Exemplaire(n_livre, n_exemplaire, etat, date_acq)
- Emprunteur(n_inscr, nom, prenom, adresse)
- Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)

(a) Titres écrits par Yves Fonctionrecurs

(b) Numéros, noms et prénoms des emprunteurs qui ont en prêt un exemplaire du livre n°10

(c) Nombre d'emprunts en cours

(d) Pour chaque livre, numéros, titres et nombre d'exemplaires en prêt

(e) Numéros et titres des livres édités depuis le 1er septembre 2003

(f) Noms, prénoms et codes des auteurs ayant écrit le plus de livres dans la bibliothèque

(g) Numéros et titres des livre encore disponibles (il reste au moins un exemplaire)

(h) Numéros et titres des livres dont tous les exemplaires ont été empruntés

(i) Numéros et titres des livres disponibles édités après le 1er septembre 2003 dont le titre contient le mot *motivation*.

(a) Titres écrits par Yves Fonctionrecurs

Auteur(<u>code_auteur</u>, nom, prenom)
Livre(<u>n_livre</u>, ISBN, titre, editeur, date_edition)
Ecrit(<u>code_auteur, n_livre</u>)

```
SELECT titre
FROM Auteur NATURAL JOIN Ecrit NATURAL JOIN Livre
WHERE Nom="Fonctionrecurs" AND prenom="Yves"
```

(b) Numéro, nom et prénom des emprunteurs qui ont en prêt un exemplaire du livre n°10

Emprunteur(n_inscr, nom, prenom, adresse)
Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)

```
SELECT DISTINCT n_inscr, nom, prenom
FROM Emprunt NATURAL JOIN Emprunteur
WHERE n_livre=10
```

(c) Nombre d'emprunts en cours

Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)

```
SELECT COUNT(*)
FROM Emprunt
```

(d) Pour chaque livre, numéro, titre et nombre d'exemplaires en prêt (I)

Livre(n_livre, ISBN, titre, editeur, date_edition)
Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)

1. Numéro et nombre d'emprunt(s) des livres empruntés au moins une fois

```
SELECT n_livre, COUNT(*) as n_empruntes
FROM Emprunt
GROUP BY n_livre
```

2. Numéro et nombre d'emprunt(s) des livres qui ne sont pas empruntés

```
SELECT n_livre, 0 as n_empruntes
FROM Livre
WHERE n_livre NOT IN (SELECT n_livre FROM Emprunt)
```

(d) Pour chaque livre, numéro, titre et nombre d'exemplaires en prêt (II)

Livre(n_livre, ISBN, titre, editeur, date_edition)
Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)

3. Union de ces deux tables

```
(SELECT n_livre, COUNT(*) as n_empruntes
FROM Emprunt
GROUP BY n_livre)

UNION

(SELECT n_livre, 0 as n_empruntes
FROM Livre
WHERE n_livre NOT IN (SELECT n_livre FROM Emprunt))
```

(d) Pour chaque livre, numéro, titre et nombre d'exemplaires en prêt (III)

Livre(<u>n_livre</u>, ISBN, titre, editeur, date_edition)
Emprunt(<u>n_livre, n_exemplaire, n_inscr, date_empr</u>)

4. NATURAL JOIN avec Livre pour récupérer le titre

```
SELECT n_livre, titre, n_empruntes
FROM
    ((SELECT n_livre, COUNT(*) as n_empruntes
    FROM Emprunt
    GROUP BY n_livre)
    UNION
    (SELECT n_livre, 0 as n_empruntes
    FROM Livre
    WHERE n_livre NOT IN (SELECT n_livre FROM Emprunt))
    NATURAL JOIN Livre
```

(e) Numéros et titres des livres édités depuis le 1er septembre 2003

Livre(n_livre, ISBN, titre, editeur, date_edition)

```
SELECT n_livre, titre
FROM Livre
WHERE date_edition>='01-09-2003'
```

(f) Nom, prénom et code des auteurs ayant écrit le plus de livres dans la bibliothèque

Auteur(<u>code_auteur</u>, nom, prenom)
Ecrit(<u>code_auteur, n_livre</u>)

```
SELECT code_auteur, nom, prenom
FROM Ecrit NATURAL JOIN Auteur
GROUP BY code_auteur
HAVING COUNT(*)=MAX(COUNT(*))
```

(g) Numéro et titre des livre encore disponibles (il reste au moins un exemplaire) (I)

Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)
Exemplaire(n_livre, n_exemplaire, etat, date_acq)
Livre(n_livre, ISBN, titre, editeur, date_edition)

1. Sélection des numéros de livres d'exemplaires qui n'ont pas été emprunté

```
SELECT n_livre
FROM Exemplaire
WHERE n_livre, n_exemplaire NOT IN
    (SELECT n_livre,n_exemplaire FROM Emprunt)
```

(g) Numéro et titre des livre encore disponible (il reste au moins un exemplaire) (II)

Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)
Exemplaire(n_livre, n_exemplaire, etat, date_acq)
Livre(n_livre, ISBN, titre, editeur, date_edition)

2. NATURAL JOIN avec Livre pour obtenir les titres

```
SELECT titre, n_livre
FROM
    (SELECT n_livre
    FROM Exemplaire
    WHERE n_livre, n_exemplaire NOT IN
        (SELECT n_livre,n_exemplaire FROM Emprunt))
    NATURAL JOIN Livre
```

(h) Numéro et titre des livres dont tous les exemplaires ont été empruntés (I)

Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)
Exemplaire(n_livre, n_exemplaire, etat, date_acq)
Livre(n_livre, ISBN, titre, editeur, date_edition)

On cherche l'inverse de la question précédente !

1. Rechercher les identifiants de livres disponibles

```
SELECT n_livre
FROM Exemplaire
WHERE n_livre, n_exemplaire NOT IN
    (SELECT n_livre,n_exemplaire FROM Emprunt)
```

(h) Numéro et titre des livres dont tous les exemplaires ont été empruntés (II)

Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)
Exemplaire(n_livre, n_exemplaire, etat, date_acq)
Livre(n_livre, ISBN, titre, editeur, date_edition)

2. Chercher les livres qui ne sont pas présents dans cette table.

```
SELECT titre, n_livre
FROM Livre
WHERE n_livre NOT IN
    (SELECT n_livre
    FROM Exemplaire
    WHERE n_livre, n_exemplaire NOT IN
        (SELECT n_livre,n_exemplaire FROM Emprunt))
```

(i) Numéro et titre des livres disponibles édités après le 1er septembre 2003 dont le titre contient le mot *motivation*.

Emprunt(n_livre, n_exemplaire, n_inscr, date_empr)
Exemplaire(n_livre, n_exemplaire, etat, date_acq)
Livre(n_livre, ISBN, titre, editeur, date_edition)

Encore très semblable à (g)...

```
SELECT titre, n_livre
FROM
    ((SELECT n_livre
    FROM Exemplaire
    WHERE n_livre, n_exemplaire NOT IN
        (SELECT n_livre,n_exemplaire FROM Emprunt))
    NATURAL JOIN Livre)
WHERE date_edition>'01-09-2003'
AND titre LIKE '%motivation%'
```