JOUEURS ET MATCHS

DONNEES

La base de données **SPORT** est constituée des trois relations suivantes :
joueurs(j_id:number, j_nom:string, j_taille:number) : données sur les joueurs
matchs(m_id:number, m_ville:string, m_date:string) : données sur les villes et dates des matchs
matchs_joueurs(mj_id:number, m_id:number, j_id:number) : données sur les équipes lors des matchs

j_id	j_nom	j_taille
1	Jean	1.75
2	Pierre	1.85
3	Lucie	1.65
4	Chloé	1.79
5	Chloé	1.64
...	...	...

Relation « joueurs »

m_id	m_ville	m_date
1	Cholet	2016-11-20
2	Saumur	2016-11-23
3	Angers	2016-11-27
4	Cholet	2016-12-01
...	...	...

Relation « matchs »

mj_id	m_id	j_id
1	1	1
2	1	3
3	1	5
4	1	8
5	2	2
6	2	3
...	...	...

« matchs_joueurs »

REQUETES (SOLUTIONS PAGE 3)

Écrire les requêtes suivantes en algèbre relationnelle puis en SQL :

1. Liste distincte des noms des joueurs
2. Classez les noms des joueurs par tailles croissantes
3. Liste des noms et tailles des joueurs mesurant plus de 1m80
4. Nombre de joueurs ayant participé au match n°1
5. Liste des noms des joueurs ayant participé au match n°1
6. Liste des noms des joueurs ayant participé à plusieurs matchs
7. Taille moyenne des joueurs par ville où se sont déroulés les matchs
8. Liste des joueurs ayant eu un match en décembre
9. Liste des joueurs dont le prénom commence par la lettre « P »
10. Liste des joueurs dont la taille est supérieure ou égale à la moyenne
11. Les équipes doivent normalement comporter 4 joueurs. Affichez la liste des villes, date et nombre de joueurs qui ne vérifiaient pas ce critère.

PROGRAMMATION PYTHON (SOLUTIONS PAGE 9)

1. Créez un programme permettant d'obtenir l'affichage des matchs pour l'ensemble des joueurs :

```
Match(s) joués par  Jean
==========                              Match(s) joués par  Pierre
Cholet le 20 novembre 2016              ==========
Angers le 27 novembre 2016              Saumur le 23 novembre 2016
Saumur le 11 décembre 2016              Angers le 15 décembre 2016
Angers le 15 décembre 2016              Angers le 16 décembre 2016
```

...

2. Créez un programme qui demande à un utilisateur d'entrer une taille en cm puis affiche tous les matchs où les joueurs ne dépassaient pas cette taille. Par exemple, en tapant 180 on obtiendra :

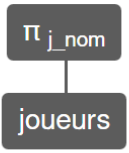
```
Je trouve 3 match(s) où les joueurs ne mesuraient pas plus de 1.8 m

Match(s) n° 1
==========
 Jean       ( 1.75 )
 Lucie      ( 1.65 )
 Chloé      ( 1.64 )
 Emilie     ( 1.72 )
Match(s) n° 3
==========
 Jean       ( 1.75 )
 Chloé      ( 1.64 )
 Chloé      ( 1.79 )
Match(s) n° 7
==========
 Emilie     ( 1.72 )
 Chloé      ( 1.79 )
 Chloé      ( 1.64 )
 Jean       ( 1.75 )
```

SOLUTION – JOUEURS ET MATCHS

REQUETES

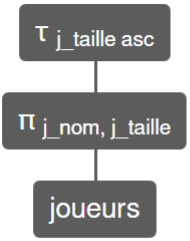
1. Liste distincte des noms des joueurs



π_{j_nom} joueurs

```
SELECT DISTINCT j_nom
FROM joueurs
```

2. Classez les noms des joueurs par tailles croissantes

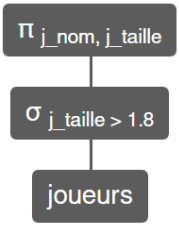


$\tau_{j\_taille\ asc}\ \pi_{j\_nom,\ j\_taille}$ (joueurs)

```
SELECT DISTINCT
j_nom,j_taille
FROM joueurs
ORDER BY j_taille ASC
```

joueurs.j_nom	joueurs.j_taille
Chloé	1.64
Lucie	1.65
…	…
Paul	1.87
Damien	1.94

3. Liste des noms et tailles des joueurs mesurant plus de 1m80

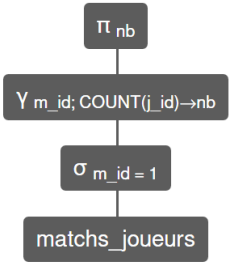


$\pi_{j\_nom,\ j\_taille}\ \sigma_{j\_taille\ >\ 1.8}$ joueurs

```
SELECT DISTINCT
j_nom,j_taille
FROM joueurs
WHERE j_taille>1.8
```

joueurs.j_nom	joueurs.j_taille
Pierre	1.85
Damien	1.94
Paul	1.87

4. Nombre de joueurs ayant participé au match n°1

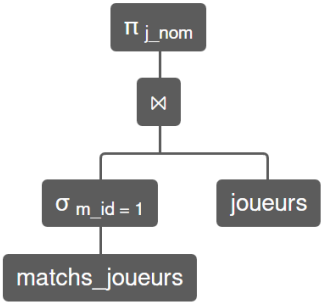


$\pi_{nb}\ \gamma_{m\_id;\ COUNT(j\_id)\rightarrow nb}\ \sigma_{m\_id\ =\ 1}$ matchs_joueurs

```
SELECT DISTINCT
COUNT(j_id) AS nb
FROM matchs_joueurs
WHERE m_id=1
GROUP BY m_id
```

nb
4

5. Liste des noms des joueurs ayant participé au match n°1



π_{j_nom} ($\sigma_{m\_id\ =\ 1}$ matchs_joueurs ⋈ joueurs)

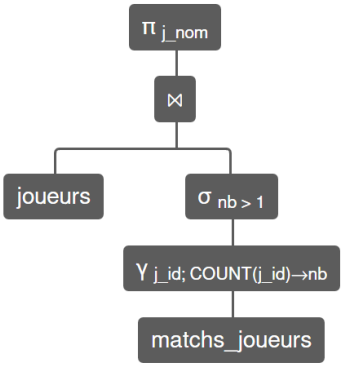
```
SELECT DISTINCT j_nom
FROM joueurs
NATURAL JOIN
matchs_joueurs
WHERE m_id=1
```

joueurs.j_nom
Jean
Lucie
Chloé
Emilie

6. Liste des noms des joueurs ayant participé à plusieurs matchs

Version 1

```
SELECT DISTINCT j_nom
FROM joueurs,
(
SELECT DISTINCT j_id, COUNT(j_id) AS nb
FROM matchs_joueurs
GROUP BY j_id
HAVING nb>1) AS Y
WHERE joueurs.j_id=Y.j_id
```



π_{j_nom} (joueurs $\bowtie$ $\sigma_{nb > 1}$ ($\gamma_{j\_id;\ COUNT(j\_id)\rightarrow nb}$ matchs_joueurs))

Version 2

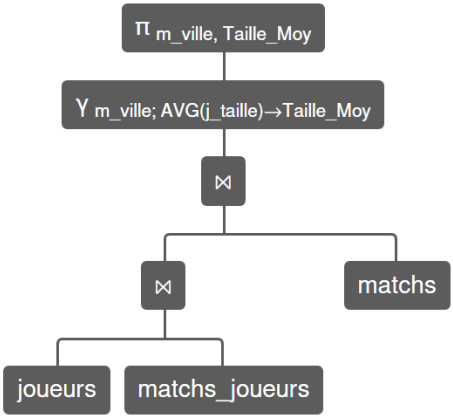
```
SELECT DISTINCT j_nom
FROM joueurs natural join matchs_joueurs
GROUP BY j_nom
HAVING Count(j_id)>1
```

joueurs.j_nom
Jean
Pierre
...
Paul
Emilie

7. Taille moyenne des joueurs par ville où se sont déroulés les matchs

Version 1

```
SELECT DISTINCT matchs.m_ville, Avg(joueurs.j_taille) AS
Taille_Moy
FROM joueurs, matchs, matchs_joueurs
WHERE joueurs.j_id = matchs_joueurs.j_id AND matchs.m_id
= matchs_joueurs.m_id
GROUP BY matchs.m_ville
```



π m_ville, Taille_Moy γ m_ville; AVG(j_taille)→Taille_Moy joueurs ⋈ matchs_joueurs ⋈ matchs

Version 2

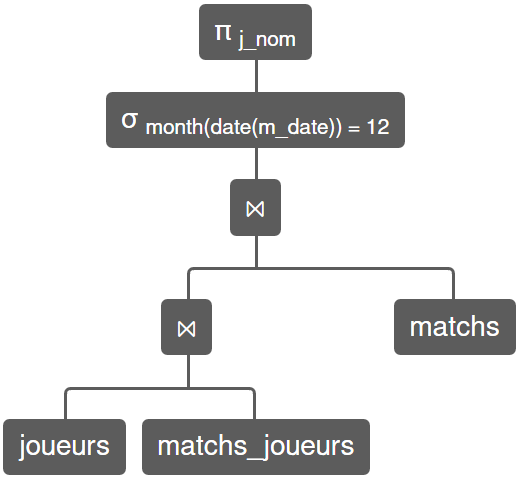
```
SELECT DISTINCT m_ville, Avg(j_taille) AS Taille_Moy
FROM joueurs natural join matchs_joueurs natural join
matchs
GROUP BY m_ville
```

matchs.m_ville	Taille_Moy
Cholet	1.7028571428571428
Angers	1.773
Saumur	1.77625

8. Liste des joueurs ayant eu un match en décembre

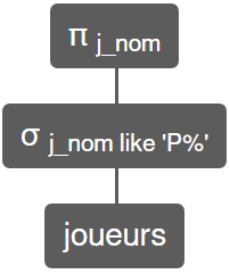
```
SELECT DISTINCT j_nom
FROM joueurs
NATURAL JOIN matchs_joueurs
NATURAL JOIN matchs
WHERE MONTH(DATE(m_date))=12
```

joueurs.j_nom
Jean
Pierre
Lucie
Chloé
Damien
Paul
Emilie



$\pi_{j\_nom}\ \sigma_{month(date(m\_date)) = 12}$ joueurs ⋈ matchs_joueurs ⋈ matchs

9. Liste des joueurs dont le prénom commence par la lettre « P »



$\pi_{j\_nom}\ \sigma_{j\_nom\ like\ 'P\%'}$ joueurs

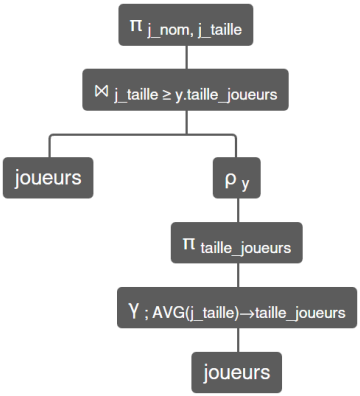
```
SELECT DISTINCT j_nom
FROM joueurs
WHERE j_nom LIKE 'P%'
```

joueurs.j_nom
Pierre
Paul

10. Liste des joueurs dont la taille est supérieure ou égale à la moyenne

Version 1

```
SELECT DISTINCT j_nom, j_taille
FROM joueurs, (SELECT DISTINCT AVG(j_taille) AS taille
FROM joueurs) AS Y
WHERE joueurs.j_taille>=Y.taille
```



π j_nom, j_taille (joueurs ⋈ j_taille ≥ y.taille_joueurs ρ y (π taille_joueurs γ ; AVG(j_taille)→taille_joueurs joueurs))

Version 2

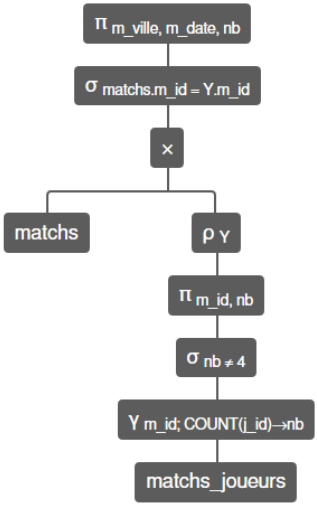
```
SELECT DISTINCT j_nom, AVG(j_taille) AS taille
FROM joueurs
GROUP BY j_id
HAVING taille>=(
SELECT AVG(j_taille)
FROM joueurs)
```

joueurs.j_nom	joueurs.j_taille
Pierre	1.85
Chloé	1.79
Damien	1.94
Paul	1.87

11. Les équipes devaient normalement comporter 4 joueurs. Affichez la liste des villes, date et nombre de joueurs qui ne vérifiaient pas ce critère.

```
SELECT DISTINCT m_ville, m_date, nb
FROM matchs, (SELECT DISTINCT m_id, COUNT(j_id) AS nb
FROM matchs_joueurs GROUP BY m_id HAVING nb<>4) AS Y
WHERE matchs.m_id=Y.m_id
```

matchs.m_ville	matchs.m_date	Y.nb
Angers	2016-11-27	3
Cholet	2016-12-03	3
Angers	2016-12-16	3



π m_ville, m_date, nb σ matchs.m_id = Y.m_id matchs × ρ Y (π m_id, nb σ nb ≠ 4 γ m_id; COUNT(j_id)→nb matchs_joueurs)

PROGRAMMATION PYTHON

1. Affichage des matchs pour l'ensemble des joueurs

```
#coding=utf-8
import sqlite3
mois=['janvier','février','mars','avril','mai','juin','juillet',
'août','septembre','octobre','novembre','décembre']
cnx=sqlite3.connect("sport.sqlite")
curseur=cnx.cursor()
curseur.execute("select j_id,j_nom FROM joueurs")
joueurs=curseur.fetchall()
for j in joueurs:
    print("\nMatch(s) joués par",j[1])
    print("="*10)
    curseur.execute("select distinct m_ville,m_date from matchs
natural join matchs_joueurs where j_id="+str(j[0])+" order by
m_date")
    matchs=curseur.fetchall()
    for m in matchs:
        amj=m[1].split("-")
        print(m[0], "le",amj[2],mois[int(amj[1])-1],amj[0])
curseur.close()
cnx.close()
```

2. Liste des matchs où les joueurs ne dépassent pas une taille imposée

```
#coding=utf-8
import sqlite3
cnx=sqlite3.connect("sport.sqlite")
curseur=cnx.cursor()
taille=int(input("Taille à ne pas dépasser (en cm) ?"))/100
curseur.execute("select distinct m_id,max(j_taille) as taille
from matchs_joueurs natural join joueurs group by m_id having
taille<"+str(taille))
matchs=curseur.fetchall()
if len(matchs)>0:
    print("Je trouve",len(matchs),"match(s) où les joueurs
mesuraient plus de",taille,"m")
    for m in matchs:
        print("Match(s) n°",m[0])
        print("="*10)
        curseur.execute("SELECT DISTINCT j_nom,j_taille FROM
joueurs NATURAL JOIN matchs_joueurs WHERE m_id="+str(m[0]))
        joueurs=curseur.fetchall()
        for j in joueurs:
            print(j[0], "(",j[1],")")
else:
    print("Je ne trouve pas de match")
curseur.close()
cnx.close()
```

LISTE D'ETUDIANTS

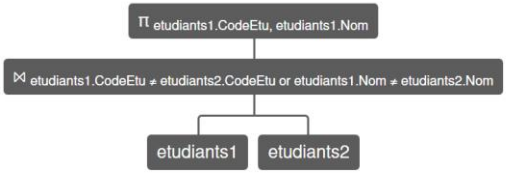
DONNEES

La base de données **ETUDIANTS** est constituée des 2 relations suivantes :
etudiants1 (N_Eleve:number, CodeEtu:number, Nom:string)
etudiants2 (N_Eleve:number, CodeEtu:number, Nom:string)

REQUETES (SOLUTIONS PAGE 13)

On cherche à savoir si les contenus des deux relations **etudiants1** et **etudiants2** sont identiques ou non.

La requête suivante propose de rechercher les noms et codes étudiants de la relation **etudiants1** qui ne sont pas identiques à la seconde relation. Montrez par un contre-exemple que cette requête n'est pas correcte.



π etudiants1.CodeEtu, etudiants1.Nom (etudiants1 ⋈ etudiants1.CodeEtu ≠ etudiants2.CodeEtu or etudiants1.Nom ≠ etudiants2.Nom (etudiants2))

1. Proposez une requête permettant de répondre à la question et d'obtenir :

etudiants1.CodeEtu	**etudiants1.Nom**
2056320	HERICH
2062936	NOEL
2063297	CARRE
2063564	DIKOUS

2. Construire une requête qui listera les différences entre les deux relations

etudiants1.CodeEtu	**etudiants1.Nom**
2063297	CARE
2063297	CARRE
..	...
2062936	NOEL
2062956	NOEL

3. Écrire les 2 requêtes précédentes en SQL

PROGRAMMATION PYTHON (SOLUTIONS PAGE 15)

Créer une fonction nommée **soundex** basée sur fr.wikipedia.org/wiki/Soundex qui admet une chaine de caractères en paramètre et telle qu'elle :

- Garde la première lettre de la chaine
- Supprime toutes les occurrences des lettres : a, e, h, i, o, u, w, y (à moins que ce ne soit la première lettre du nom) ainsi que les voyelles accentuées.
- Attribue une valeur numérique aux lettres restantes de la manière suivante :
  1 = B, P
  2 = C, K, Q
  3 = D, T
  4 = L
  5 = M, N
  6 = R
  7 = G, J
  8 = X, Z, S
  9 = F, V
- Renvoie les 4 premiers caractères complétés éventuellement par des 0.

Remarque : On supposera ici que les noms ne contiennent pas de consonnes accentuées et qu'ils ne commencent pas par une voyelle accentuée.

Ecrire un programme qui demande à un utilisateur de taper le nom exact ou approximatif d'un étudiant et qui donne en retour la liste des noms et numéros d'étudiants susceptibles de correspondre.

```
>>>
BOUCHE est peut-être :
BEAUCH ( 2062826 )
BOUCHE ( 2062950 )
BOUCHI ( 2063218 )
```

Remarque : SOUNDEX existe en SQL mais n'est pas installée par défaut sous SQLite.

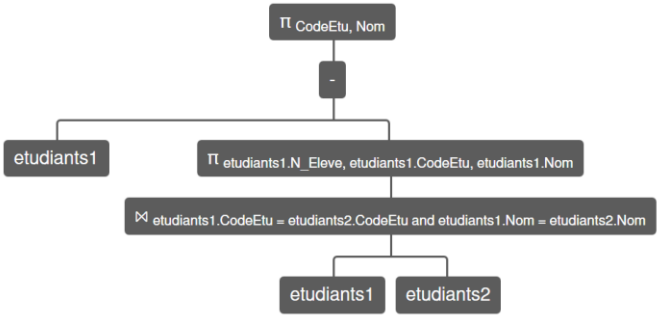
```
SELECT CodeEtu, Nom
FROM etudiants1
WHERE SOUNDEX(Nom) = SOUNDEX('BOUCHE')
```

SOLUTION – LISTE D'ETUDIANTS

REQUETES

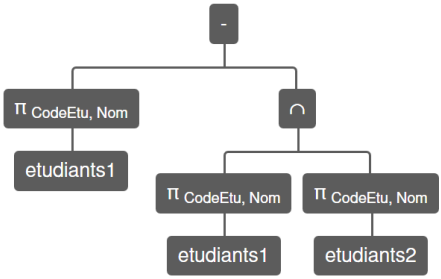
1. Liste des noms et codes étudiants de **etudiants1** qui ne sont pas identiques à la seconde relation

Version 1



$\pi_{CodeEtu, Nom}$ (etudiants1 - $\pi_{etudiants1.N\_Eleve, etudiants1.CodeEtu, etudiants1.Nom}$ (etudiants1 $\bowtie_{etudiants1.CodeEtu = etudiants2.CodeEtu \text{ and } etudiants1.Nom = etudiants2.Nom}$ (etudiants2)))

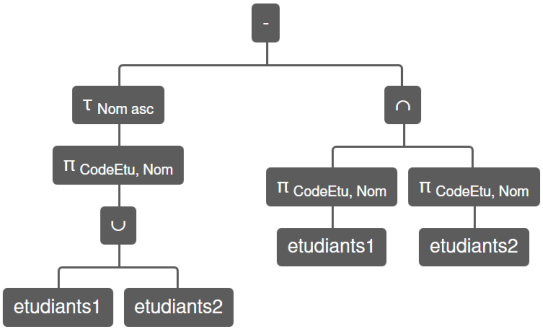
Version 2



$\pi_{CodeEtu, Nom}$ etudiants1 - $\pi_{CodeEtu, Nom}$ etudiants1 $\cap$ $\pi_{CodeEtu, Nom}$ etudiants2

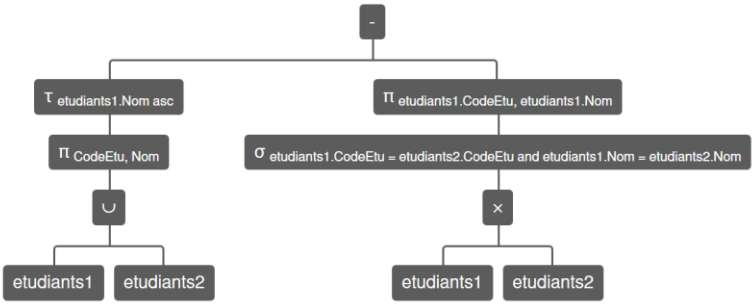
2. Liste des différences entre les deux relations

Version 1



τ Nom asc π CodeEtu, Nom (etudiants1 ∪ etudiants2) - (π CodeEtu, Nom etudiants1 ∩ π CodeEtu, Nom etudiants2)

Version 2



τ etudiants1.Nom asc π CodeEtu, Nom (etudiants1 ∪ etudiants2) - π etudiants1.CodeEtu, etudiants1.Nom σ etudiants1.CodeEtu = etudiants2.CodeEtu and etudiants1.Nom = etudiants2.Nom (etudiants1 × etudiants2)

3. Versions SQL

Requête 1

```
SELECT DISTINCT CodeEtu,Nom
FROM etudiants1 EXCEPT
SELECT DISTINCT etudiants1.CodeEtu,etudiants1.Nom
FROM etudiants1, etudiants2
WHERE etudiants1.CodeEtu=etudiants2.CodeEtu AND
etudiants1.Nom=etudiants2.Nom
```

Requête 2

```
SELECT DISTINCT CodeEtu,Nom
FROM etudiants1 UNION
SELECT DISTINCT CodeEtu,Nom
FROM etudiants2 EXCEPT
SELECT DISTINCT etudiants1.CodeEtu,etudiants1.Nom
FROM etudiants1, etudiants2
WHERE etudiants1.CodeEtu=etudiants2.CodeEtu AND
etudiants1.Nom=etudiants2.Nom
ORDER BY Nom ASC
```

PROGRAMMATION PYTHON

```
#coding=utf-8
def soundex(nom):
    digits = '01230970072455012683090808'
    snd=nom[0]
    for c in nom[1:]:
        if "A" <= c <= "Z":
            snd+=digits[ord(c)-ord('A')]
    snd=snd.replace('0','')
    return (snd+"0"*4)[:5]

import sqlite3
cnx=sqlite3.connect("etudiants.sqlite")
curseur=cnx.cursor()
etudiant=input("Nom approximatif ?").upper()
recherche=soundex(etudiant)
print(etudiant,"est peut-être :")
curseur.execute("select Nom FROM etudiants1")
joueurs=curseur.fetchall()
for j in joueurs:
    s=soundex(j[0])
    if s==recherche:
        print(j[0],'(',j[1],')')
curseur.close()
cnx.close()
```

STAGES

DONNEES

La base de données **STAGE** est constituée des deux relations suivantes :
stages(N_Stagiaire:number, Nom:string, Prenom:string, Debut:date, semaines:number, Labo:string) : liste des stages avec nom, prénom, date du début du stage, nombre de semaines du stage et laboratoire d'accueil.
laboratoires(Labo:string, Maxi:number) : liste des laboratoires et capacité d'accueil maximale de stagiaires.

REQUETES (SOLUTIONS PAGE 19)

Écrire les requêtes suivantes en algèbre relationnelle puis en SQL :

1. Nombre total de semaines de stages par laboratoire
2. Liste des noms et prénoms des stagiaires avec dates de début et de fin de stage. On classera le tableau par ordre chronologique de début de stage
3. Obtenir la liste ci-dessous permettant de visualiser les stages se chevauchant pour un même laboratoire. Par exemple LEKA a un stage de 2 semaines dans le laboratoire A à partir du 1/7/2015, il chevauchera donc celui de ZIHOUNE qui commence lui le 8/7/2015.

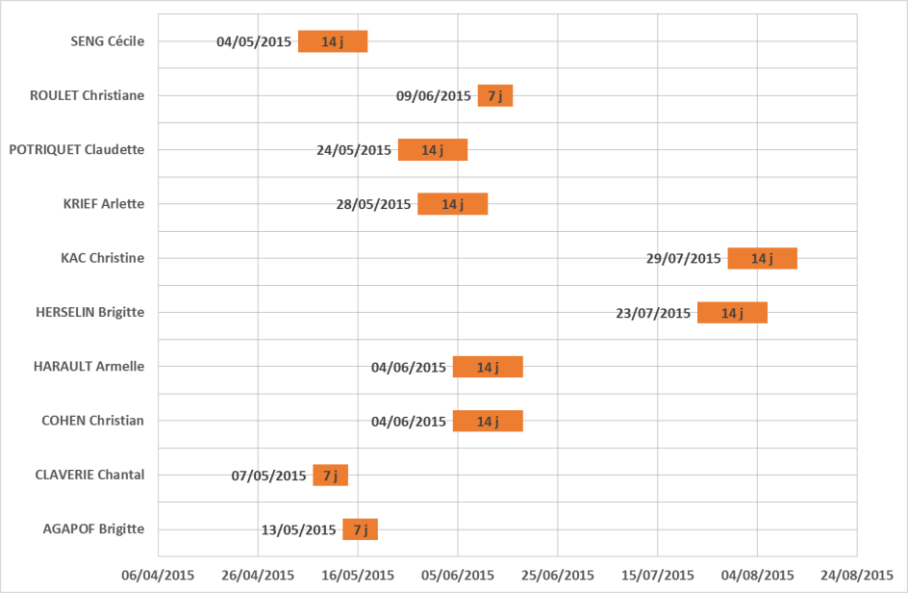
s1.Debut	**s1.Semaines**	**s1.Nom**	**s1.Prenom**	**s2.Nom**	**s2.Prenom**	**s1.Labo**
2015-07-01	2	LEKA	Bernadette	ZIHOUNE	Christiane	A
2015-07-08	2	ZIHOUNE	Christiane	LEKA	Bernadette	A
2015-06-02	1	CHHUOR	Anne-Marie	SAPIENCE	Alain	B
2015-06-02	1	CHHUOR	Anne-Marie	GONDOUIN	Bernard	B
2015-06-05	2	SAPIENCE	Alain	GIRON	Anne-Marie	B
2015-06-05	2	SAPIENCE	Alain	CHHUOR	Anne-Marie	B
2015-06-05	2	SAPIENCE	Alain	GONDOUIN	Bernard	B
2015-06-07	1	GONDOUIN	Bernard	SAPIENCE	Alain	B
2015-06-07	1	GONDOUIN	Bernard	GIRON	Anne-Marie	B
…	…	…	…	…	…	…

4. Obtenir, par laboratoire, la liste des stagiaires où l'on dépassera le nombre maximum autorisé

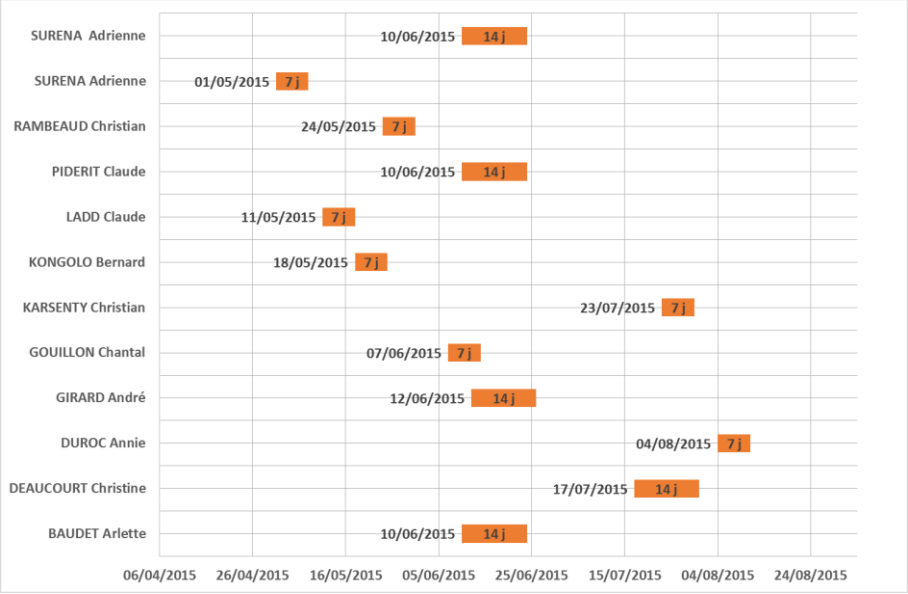
laboratoires.Labo	**s1.Nom**	**nb**
D	AGAPOF	2
D	CLAVERIE	2
D	COHEN	4
…	…	…
D	SENG	2
E	PIDERIT	4
E	SURENA	4

AGAPOF, CLAVERIE et SENG seront dans le laboratoire D qui est limité à 2 stagiaires

De même 5 stagiaires seront dans le laboratoire E qui est limité à 4



Visualisation des stages pour le laboratoire D



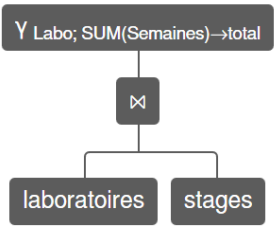
Visualisation des stages pour le laboratoire E

PROGRAMMATION PYTHON (SOLUTIONS PAGE 21)

SOLUTION – STAGES

REQUETES

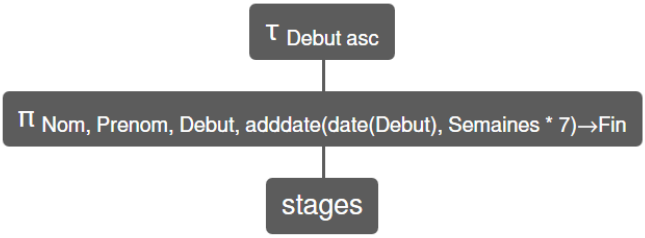
1. Nombre total de semaines de stages par laboratoire



γ Labo; SUM(Semaines)→total (laboratoires ⋈ stages)

```
SELECT DISTINCT Labo, SUM(Semaines) AS total
FROM stages
NATURAL JOIN laboratoires
GROUP BY Labo
ORDER BY Labo
```

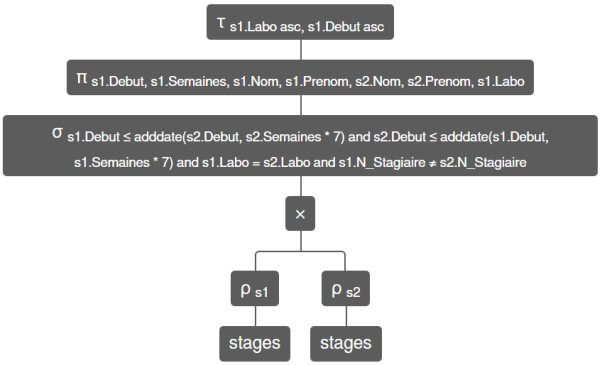
2. Liste des noms et prénoms des stagiaires avec dates de début et de fin de stage. On classera le tableau par ordre chronologique de début de stage.



τ Debut asc π Nom, Prenom, Debut, adddate(date(Debut), Semaines * 7)→Fin stages

```
SELECT DISTINCT Nom, Prenom, Debut,
ADDDATE(Debut,Semaines*7) AS Fin
FROM stages
ORDER BY Debut
```

3. Liste des chevauchements de stages

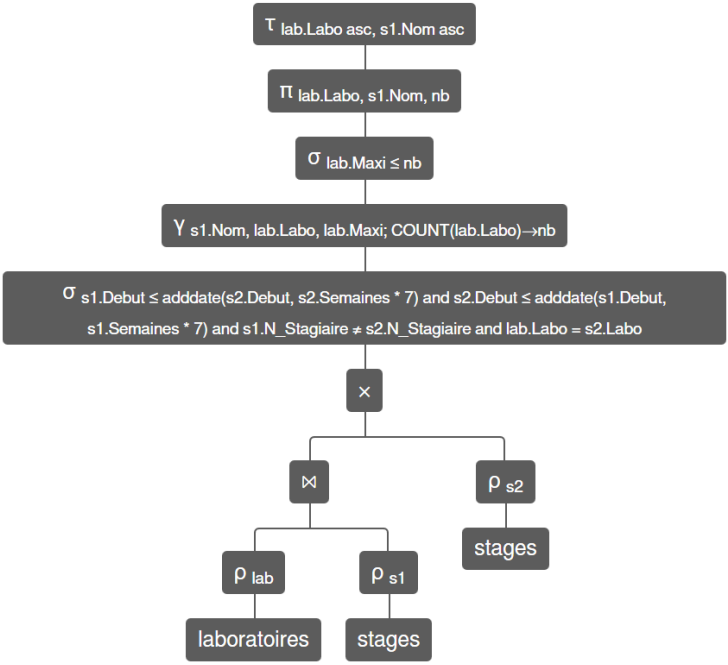


τ s1.Labo asc, s1.Debut asc π s1.Debut, s1.Semaines, s1.Nom, s1.Prenom, s2.Nom, s2.Prenom, s1.Labo σ s1.Debut ≤ adddate(s2.Debut, s2.Semaines * 7) and s2.Debut ≤ adddate(s1.Debut, s1.Semaines * 7) and s1.Labo = s2.Labo and s1.N_Stagiaire ≠ s2.N_Stagiaire ρ s1 stages × ρ s2 stages

```
SELECT DISTINCT s1.Debut,s1.Semaines,s1.Nom, s1.Prenom,
s2.Nom, s2.Prenom, s1.Labo
FROM stages AS s1, stages AS s2
WHERE
s1.Debut<= ADDDATE(s2.Debut,s2.Semaines*7) AND
s2.Debut<= ADDDATE(s1.Debut,s1.Semaines*7) AND
s1.Labo=s2.Labo AND s1.N_Stagiaire <> s2.N_Stagiaire
ORDER BY s1.Labo ASC, s1.Debut ASC
```

4. Dépassement des capacités

```
SELECT DISTINCT lab.Labo, s1.Nom, COUNT(lab.Labo) AS nb
FROM laboratoires AS lab
NATURAL JOIN stages AS s1, stages AS s2
WHERE
s1.Debut<= ADDDATE(s2.Debut,s2.Semaines*7) AND
s2.Debut<= ADDDATE(s1.Debut,s1.Semaines*7) AND
s1.N_Stagiaire<>s2.N_Stagiaire AND lab.Labo=s2.Labo
GROUP BY s1.Nom,lab.Labo, lab.Maxi
HAVING lab.Maxi<=nb
ORDER BY lab.Labo, s1.Nom
```



τ lab.Labo asc, s1.Nom asc π lab.Labo, s1.Nom, nb σ lab.Maxi ≤ nb γ s1.Nom, lab.Labo, lab.Maxi; COUNT(lab.Labo)→nb σ s1.Debut ≤ adddate(s2.Debut, s2.Semaines * 7) and s2.Debut ≤ adddate(s1.Debut, s1.Semaines * 7) and s1.N_Stagiaire ≠ s2.N_Stagiaire and lab.Labo = s2.Labo (ρ lab laboratoires ⋈ ρ s1 stages × ρ s2 stages)

laboratoires.Labo	**s1.Nom**	**nb**
D	AGAPOF	2
D	SENG	2
D	CLAVERIE	2
D	ROULET	3
...	...	...
E	GOUILLON	4
E	PIDERIT	4

PROGRAMMATION PYTHON

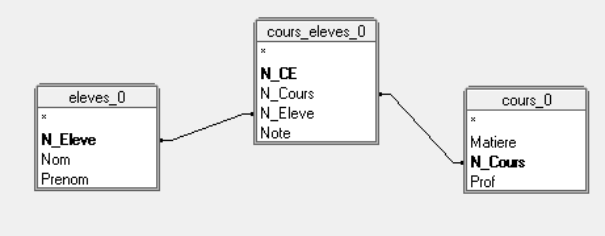
ECOLE

DONNEES

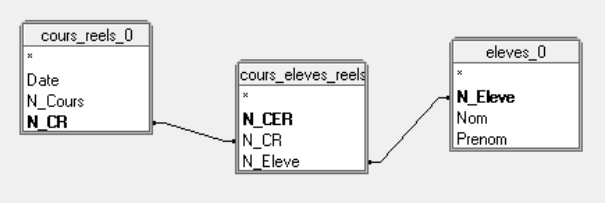
La base de données relationnelles **ECOLE** est constituée des cinq relations suivantes :
eleves(N_Eleve:number, Nom:string, Prenom:string) : Données sur les élèves
cours(N_Cours:number, Matiere:string, Prof:string) : Donnée sur les cours
cours_eleves(N_CE:number, N_Cours:number, N_Eleve:number, Note:number) : Inscriptions des élèves aux différents cours et note obtenue dans ce cours
cours_reels(N_CR:number, N_Cours:number, Date:string) : Cours réellement donnés avec la date de la séance
cours_eleves_reels(N_CER:number, N_Eleve:number, N_CR:number) : Pointage des élèves présents aux différents cours donnés

Remarques :

- Un élève peut n'être inscrit à aucun cours (cause d'abandon par exemple)
- Un élève peut suivre ponctuellement un cours sans y être inscrit. Il ajoute alors son nom sur la feuille d'émargement et sera pointé par le secrétariat



Inscription des élèves et cours proposés



Cours réels et pointage des présents

REQUETES (SOLUTIONS PAGE 24)

Écrire les requêtes suivantes en algèbre relationnelle puis en SQL :

1. Liste des noms et prénoms des élèves ayant suivi au moins un cours de maths
2. Quelle différence faites-vous entre :

π N_Eleve eleves - (π N_Eleve cours_eleves $\cap$ π N_Eleve cours_eleves_reels)

et

π N_Eleve (π N_Cours cours_reels $\bowtie$ cours_eleves) - π N_Eleve cours_eleves_reels

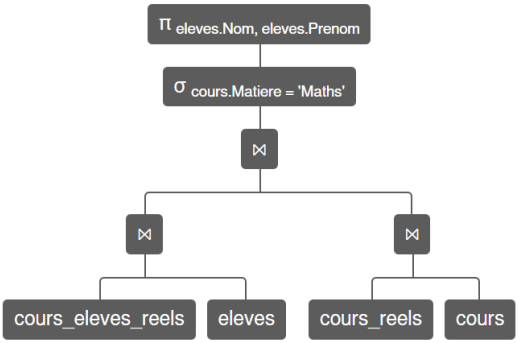
3. Liste des élèves ayant manqué au moins un cours
4. Liste des élèves n'ayant manqué aucun des cours auxquels ils sont inscrits (Attention au cas des élèves ayant abandonné la formation)
5. Liste des élèves ayant suivi des cours auxquels ils n'étaient pas inscrits, on précisera alors le numéro du cours et la matière enseignée (Par exemple SURENA a suivi le cours n°3 de Maths sans y être inscrite)

eleves.Nom	**eleves.Prenom**	**cours.Matiere**	**cours.N_Cours**
SURENA	Adrienne	Maths	3
GIRARD	André	Français	1
MARTI	Anne	Français	7
FRANÇOIS	Anne-Marie	Français	1
...	...	...	
DUROC	Annie	Français	1
DUROC	Annie	Français	7
BAUDET	Arlette	Français	1
SCHUSTER	Bernadette	Français	1

PROGRAMMATION PYTHON (SOLUTION PAGE 27)

REQUETES

1. Liste des noms et prénoms des élèves ayant suivi au moins un cours de maths



$\pi_{\text{eleves.Nom, eleves.Prenom}}\ \sigma_{\text{cours.Matiere = 'Maths'}}\ (\text{cours\_eleves\_reels} \bowtie \text{eleves} \bowtie (\text{cours\_reels} \bowtie \text{cours}))$

```
SELECT DISTINCT eleves.Nom, eleves.Prenom
FROM cours_eleves_reels
NATURAL JOIN eleves
NATURAL JOIN cours_reels
NATURAL JOIN cours
WHERE cours.Matiere='Maths'
```

eleves.Nom	**eleves.Prenom**
HARAULT	Armelle
BAUDET	Arlette
TANG	Armelle
FRANÇOIS	Anne-Marie
...	...
CHHUOR	Anne-Marie
CHARDON	Annick

2. Quelle différence faites-vous entre :

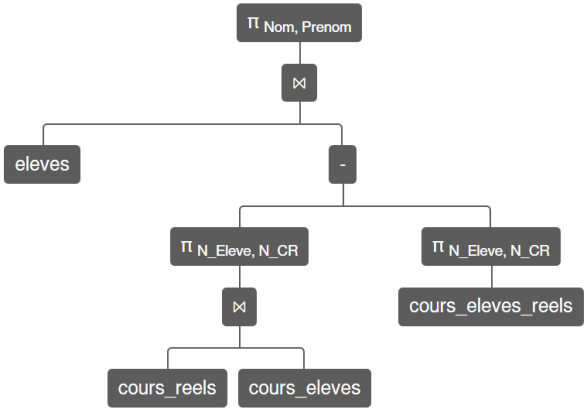
π N_Eleve eleves - (π N_Eleve cours_eleves ∩ π N_Eleve cours_eleves_reels)

Cette requête renvoie non seulement les élèves n'ayant suivi aucun cours mais également ceux qui ont abandonné (comme l'élève n°21)

π N_Eleve (π N_Cours cours_reels ⋈ cours_eleves) - π N_Eleve cours_eleves_reels

Cette fois-ci on récupère la liste des élèves qui auraient dû suivre les cours réellement donnés puis on soustrait l'ensemble des présents.

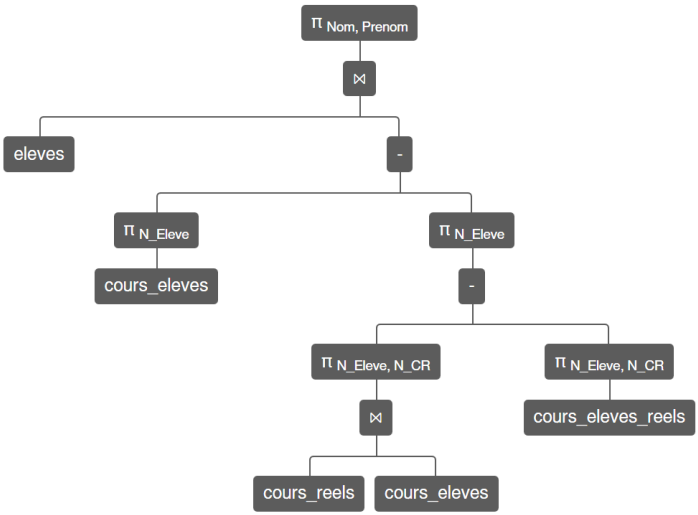
3. Liste des élèves ayant manqué au moins un cours



π Nom, Prenom (eleves ⋈ (π N_Eleve, N_CR (cours_reels ⋈ cours_eleves) - π N_Eleve, N_CR cours_eleves_reels))

```
SELECT DISTINCT Nom,Prenom
FROM eleves
NATURAL JOIN (
SELECT DISTINCT N_Eleve
FROM (
SELECT DISTINCT N_Eleve,N_CR
FROM cours_reels
NATURAL JOIN cours_eleves except
SELECT DISTINCT N_Eleve,N_CR
FROM cours_eleves_reels) AS absents) AS numero
```

4. Liste des élèves n'ayant manqué aucun des cours auxquels ils sont inscrits

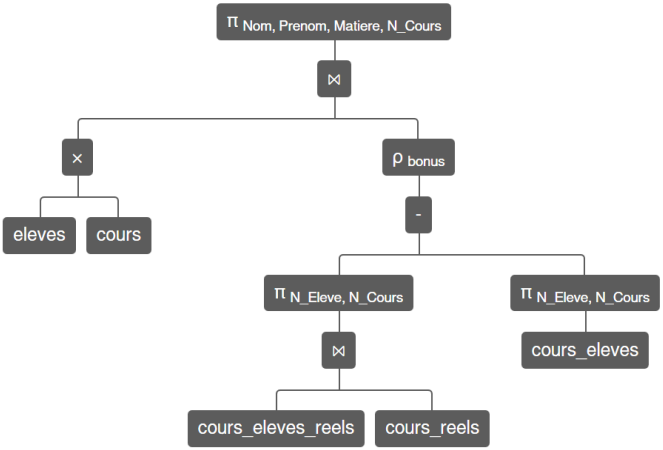


$\pi_{Nom, Prenom}$ (eleves ⋈ (π_{N_Eleve} cours_eleves - π_{N_Eleve} (π_{N_Eleve, N_CR} (cours_reels ⋈ cours_eleves) - π_{N_Eleve, N_CR} cours_eleves_reels)))

```
SELECT DISTINCT Nom,Prenom
FROM eleves
NATURAL JOIN (
SELECT DISTINCT N_Eleve
FROM cours_eleves except (
SELECT DISTINCT N_Eleve
FROM (
SELECT DISTINCT N_Eleve,N_CR
FROM cours_reels
NATURAL JOIN cours_eleves except
SELECT DISTINCT N_Eleve,N_CR
FROM cours_eleves_reels) AS absents)) AS numero
```

eleves.N_Eleve	eleves.Nom	eleves.Prenom
2	SAPIENCE	Alain
4	GIRARD	André

5. Liste des élèves ayant suivi des cours auxquels ils n'étaient pas inscrits, on précisera alors le numéro du cours et la matière enseignée



$\pi_{Nom, Prenom, Matiere, N\_Cours}$ (eleves × cours ⋈ ρ_{bonus} (π_{N_Eleve, N_Cours} (cours_eleves_reels ⋈ cours_reels) - π_{N_Eleve, N_Cours} cours_eleves))

```
SELECT DISTINCT Nom,Prenom,Matiere,N_Cours
FROM eleves, cours
NATURAL JOIN (
SELECT DISTINCT N_Eleve,N_Cours
FROM cours_eleves_reels
NATURAL JOIN cours_reels except
SELECT DISTINCT N_Eleve,N_Cours
FROM cours_eleves) AS bonus
```

PROGRAMMATION PYTHON