Exercice du cours : requête renvoyant l'atelier avec le plus de places libres.

```
SELECT nom FROM atelier WHERE nbplace-nbinscrit =
(SELECT MAX(pllibres) FROM (SELECT nbplace-nbinscrit AS pllibres
FROM atelier) AS temp)
```

Remarque : on peut utilizer WITH pour une requête un peu plus conviviale :

```
WITH atelier2 AS (SELECT *,nbplace-nbinscrit AS pllibres FROM atelier)
SELECT nom FROM atelier2 WHERE pllibres =
        (SELECT MAX(pllibres) FROM atelier2)
```

WITH permet de créer une nouvelle table qui reprend la table atelier en rajoutant une colonne correspondant aux places libres.

Corrigé Exercice

1. (Abonne_ID, Livre_ID, DateEmprunt)

2. a. $\pi_{Nom,Prenom}(\sigma_{Ville=Bayonne}(Abonne))$

```
SELECT Nom, Prenom FROM Abonne WHERE Ville = 'Bayonne'
```

b. $\pi_{Nom,Prenom}(\sigma_{CodePostal\ LIKE\ '64\%'}(Abonne))$

```
SELECT Nom, Prenom FROM Abonne WHERE CodePostal LIKE '64%'
```

c. $\pi_{Abonne\_ID}(\sigma_{DateRetourReel>DateRetourPrevu}(Emprunt))$

```
SELECT Abonne_ID FROM Emprunt WHERE
                        DateRetourReel > DateRetourPrevu
```

d. $\pi_{Nom}(\sigma_{DateRetourReel>DateRetourPrevu}(Abonne \bowtie_{Abonne.Abonne\_ID = Emprunt.Abonne\_ID} Emprunt))$

```
SELECT Nom FROM Abonne JOIN Emprunt
        ON Abonne.Abonne_ID = Emprunt.Abonne_ID
        WHERE DateRetourReel > DateRetourPrevu
```

e. $\pi_{Titre}(\sigma_{DateRetourReel\ IS\ NULL}(Livre \bowtie_{Livre.Livre\_ID = Emprunt.Livre\_ID} Emprunt))$

```
SELECT Titre FROM Livre JOIN Emprunt
        ON Livre.Livre_ID = Emprunt.Livre_ID
        WHERE DateRetourReel IS NULL
```

f. $\pi_{Nom}(\sigma_{Livre.Titre = 'Perceval'}(Abonn \bowtie_{Abonne.Abonne\_ID = Emprunt.Abonne\_ID} Emprunt \bowtie_{Emprunt.Livre\_ID = Livre.Livre\_ID} Livre))$

```
SELECT Nom FROM Abonne
        JOIN Emprunt ON Abonne.Abonne_ID = Emprunt.Abonne_ID
        JOIN Livre ON Livre.Livre_ID = Emprunt.Livre_ID
        WHERE Livre.Titre = 'Perceval'
```

g.
```
SELECT * FROM Abonne
   WHERE NOT EXISTS (SELECT * FROM Emprunt
                        WHERE Abonne.Abonne_ID = Emprunt.Abonne_ID)
```

ou

```
SELECT * FROM Abonne
   WHERE Abonne_ID NOT IN (SELECT Abonne_ID FROM Emprunt)
```

h.
```
SELECT Nom FROM Abonne WHERE Abonne_ID NOT IN
        (SELECT Abonne_ID FROM Emprunt
        WHERE DateRetourReel > DateRetourPrevu)
```

ou

```
SELECT Nom FROM Abonne WHERE NOT EXISTS
        (SELECT * FROM Emprunt AS E
        WHERE Abonne.Abonne_ID = E.Abonne_ID
        AND E.DateRetourReel > E.DateRetourPrevu)
```

i.
```
SELECT Nom FROM Abonne WHERE Abonne_ID NOT IN
        (SELECT Abonne_ID FROM Emprunt
        WHERE DateRetourReel <= DateRetourPrevu)
```

ou

```
SELECT Nom FROM Abonne WHERE NOT EXISTS
        (SELECT * FROM Emprunt
        WHERE Abonne.Abonne_ID = Emprunt.Abonne_ID
        AND Emprunt.DateRetourReel <= Emprunt.DateRetourPrevu)
```

j. SELECT Nom, Prenom FROM Abonne
JOIN Emprunt ON Abonne.Abonne_ID = Emprunt.Abonne_ID
WHERE Ville = 'Bayonne' AND DateRetourPrevu<DateRetourReel

k. SELECT Abonne_ID FROM Abonne WHERE DateNaissance = (SELECT Min(DateNaissance) FROM Abonne)
ou
SELECT Abonne_ID FROM Abonne ORDER BY DateNaissance LIMIT 1

l. SELECT MAX(DateRetourReel-DateRetourPrevu) FROM Emprunt

Certains environnement SQL ne permettent pas d'effectuer de calculs à l'intérieur de la fonction MAX, alors on doit créer la table Temp avec la colonne retard. Ce n'est pas nécessaire aux concours sauf mention contraire.

SELECT MAX(retard) FROM (SELECT DateRetourReel-DateRetourPrevu AS retard FROM Emprunt) AS Temp

m. SELECT Abonne_ID FROM Emprunt WHERE DateRetourReel-DateRetourPrevu = (SELECT MAX(DateRetourReel-DateRetourPrevu) FROM Emprunt)

n. SELECT Abonne_ID, AVG(DateRetourReel-DateEmprunt) FROM Emprunt GROUP BY Abonne_ID

o. SELECT AVG(DateRetourReel-DateRetourPrevu) FROM Emprunt WHERE DateRetourReel>DateRetourPrevu

p. SELECT Nom
FROM Abonne
JOIN Emprunt ON Abonne.Abonne_ID = Emprunt.Abonne_ID
WHERE DateRetourReel>DateRetourPrevu
GROUP BY Abonne_ID
HAVING AVG(DateRetourReel-DateRetourPrevu) > 14

q. SELECT Titre
FROM Livre
JOIN Emprunt USING (Livre_ID)
JOIN Abonne USING (Abonne_ID)
WHERE Nom = 'Duchemin'
INTERSECT
SELECT Titre
FROM Livre
JOIN Emprunt USING (Livre_ID)
JOIN Abonne USING (Abonne_ID)
WHERE Nom = 'Lamaison'

r. On pourrait rajouter à la requête suivante le fait que le livre n'a été emprunté que deux fois, mais cela ne marchera pas si Lamaison ou Duchemin ont emprunté plusieurs fois un livre :
SELECT Titre
FROM Livre
JOIN Emprunt USING (Livre_ID)
JOIN Abonne USING (Abonne_ID)
WHERE Nom = 'Duchemin'
INTERSECT
SELECT Titre
FROM Livre
JOIN Emprunt USING (Livre_ID)
JOIN Abonne USING (Abonne_ID)
WHERE Nom = 'Lamaison'
INTERSECT
SELECT Titre
FROM Livre
JOIN Emprunt USING (Livre_ID)
GROUP BY Titre
HAVING Count(*) = 2

Une version plus astucieuse (mais qui pose le même problème) :
SELECT Titre

```
FROM Livre
JOIN Emprunt USING (Livre_ID)
JOIN Abonne USING (Abonne_ID)
GROUP BY Titre
HAVING Count(*) = 2
      AND MIN(Nom) = 'Duchemin'
      AND MAX(Nom) = 'Lamaison'
```

Une solution qui marche :

```
SELECT Titre
FROM Livre
JOIN Emprunt USING (Livre_ID)
JOIN Abonne USING (Abonne_ID)
WHERE Nom = 'Duchemin'
   INTERSECT
SELECT Titre
FROM Livre
JOIN Emprunt USING (Livre_ID)
JOIN Abonne USING (Abonne_ID)
WHERE Nom = 'Lamaison'
   EXCEPT
SELECT Titre
FROM Livre
JOIN Emprunt USING (Livre_ID)
JOIN Abonne USING (Abonne_ID)
WHERE Nom <> 'Lamaison'
    AND Nom <> 'Duchemin'
```

s. `SELECT * FROM Abonne WHERE nom BETWEEN 'A' and 'I'`

t.

```
SELECT Auteur
FROM Livre
WHERE EXISTS
   (SELECT 1
   FROM Abonne
   WHERE Livre.Auteur LIKE CONCAT('%', Abonne.Nom', %')
      AND Livre.Auteur LIKE CONCAT('%', Abonne.Prenom', %') )
```

```
SELECT Auteur
FROM Livre, Abonne
WHERE Livre.Auteur LIKE CONCAT('%', Abonne.Nom', %')
    AND Livre.Auteur LIKE CONCAT('%', Abonne.Prenom', %')
```

3. Il s'agit de la division cartésienne de la table $\pi_{\text{Livre\_ID,Abonne\_ID}}$(Emprunt) par la table $\pi_{\text{Livre\_ID}}$(Livre).
$R/R' = \{s / \forall t \in R', (t, s) \in R\}$.

```
SELECT DISTINCT Abonne_ID
FROM Emprunt AS E1
WHERE NOT EXISTS
        (SELECT *
         FROM (SELECT Livre_ID FROM Livre) AS L1
         WHERE NOT EXISTS
                (SELECT *
                 FROM Emprunt AS E2
                 WHERE E1.Abonne_ID = E2.Abonne_ID
                 AND E2.Livre_ID = L1.Livre_ID)))
```

Explications : Si un abonné a emprunté tous les livres, pour chaque livre de L1, il va exister un enregistrement correspondant à la dernière requête, et donc la requête (SELECT * FROM (SELECT Livre_ID….)) ne contient aucun enregistrement, donc la requête principale va renvoyer le nom de cet abonné.
A contrario, s'il existe un livre que l'abonné n'a pas emprunté (il existe un livre pour lequel il n'existe pas de lien entre le livre et l'abonné), la requête (SELECT * FROM (SELECT Livre_ID….)) va renvoyer l'enregistrement correspondant, donc l'abonné ne va pas être sélectionné par la requête principale.

Sujet X 2016

Q16

```
SELECT id2 FROM LIENS WHERE id1 = x
```

Q17

```
SELECT nom,prenom FROM INDIVIDUS JOIN LIENS ON INDIVIDUS.id = LIENS.id2
WHERE id1 = x
```

Q18

```
SELECT DISTINCT id1 FROM LIENS WHERE id2 IN (SELECT id2 FROM LIENS WHERE
id1 = x)
```

ou

```
SELECT L1.id1 FROM LIENS AS L1
JOIN LIENS AS L2 ON L1.id2 = L2.id1
WHERE L2.id2 = x
```

Cette jointure crée une table à 3 colonnes avec des individus (L1.id1) en colonne 1, leurs amis en colonne 2 (qui représente L1.id2 = L2.id1) et les amis des amis en colonne 3 (L2.id2).

Centrale 2016

I.A.

```
SELECT COUNT(*) FROM vol WHERE jour = '2016-05-02' AND heure <
'12:00'
```

I.B.

```
SELECT id_vol FROM vol JOIN aeroport ON id_aero = depart
 WHERE ville = 'Paris' AND jour = '2016-05-02'
```

I.C. Cette requête renvoie la liste des numéros de tous les vols domestiques français (partant de France et arrivant en France) décollant le 2 mai 2016.

I.D.

```
SELECT v1.id_vol, v2.id_vol FROM vol AS v1 JOIN vol AS v2 ON
v1.depart = v2.arrivee AND v1.arrivee = v2.depart AND v1.jour = v2.jour
AND v1.niveau = v2.niveau WHERE v1.jour = '2016-05-02' AND
v1.id_vol < v2.id_vol (pour éliminer les doublons)
```

Concours à venir ?

1.

```
SELECT id, name , cost,
  (SELECT AVG(cost) FROM mobile),
  cost-(SELECT AVG(cost) FROM mobile)
 FROM mobile;
```

Remarque : la requête suivante ne fonctionne pas car elle renvoie un seul enregistrement :

```
SELECT id, name , cost, AVG(cost), cost- AVG(cost)
  FROM mobile;
```

S'il n'y a pas de GROUP BY, l'utilisation d'une fonction d'agrégation dans le SELECT renvoie un seul enregistrement.

2.

```
SELECT name FROM mobile
  WHERE cost/(SELECT AVG(cost) FROM mobile)<1.1
  AND cost/(SELECT AVG(cost) FROM mobile)>0.9
```