```sql
-- POUR S'ENTRAINER
-- 1
SELECT * FROM etudiant;

-- 2
-- π_(nom,prenom)(etudiant)
SELECT nom, prenom FROM etudiant;

-- 3
SELECT DISTINCT ville FROM etudiant;

-- 4
-- π_(nom,prenom)(σ_CP=69380(etudiant))
SELECT nom, prenom FROM etudiant
       WHERE CP = 69380;

-- 5
SELECT nom, telephone FROM etudiant
       WHERE ville = 'Lyon' OR ville = 'Villeurbanne';

-- 6
SELECT nom, prenom FROM etudiant
       WHERE telephone LIKE '07%';
       -- LIKE : sur les chaînes de caractères
       --
-- ================
-- 7
SELECT etudiant.nom FROM etudiant, prof
       WHERE prof.id = etudiant.id_prof_principal
       AND prof.bureau = 'B014';

-- 8
-- π_etudiant.nom(σ_prof.bureau='B014'(etudiant ⋈_prof.id=etudiant.id_prof_principal prof))
SELECT etudiant.nom FROM etudiant
       JOIN prof ON prof.id = etudiant.id_prof_principal
       WHERE prof.bureaU = 'B014';

-- 9
SELECT COUNT(*) FROM etudiant
       JOIN prof ON prof.id = etudiant.id_prof_principal
       WHERE prof.discipline = 'physique';

-- ================
-- 10
SELECT COUNT(*) FROM notes
       WHERE langues > 10;

-- 11
SELECT classe, AVG(math) FROM notes
       JOIN etudiant ON notes.id_etudiant = etudiant.id
       GROUP BY classe;

-- 12
SELECT classe, COUNT(*) FROM notes
       JOIN etudiant ON notes.id_etudiant = etudiant.id
       WHERE si <= 10
       GROUP BY classe;

-- 13
SELECT MAX(info) FROM notes;

-- 14
SELECT nom FROM etudiant
       JOIN notes ON notes.id_etudiant = etudiant.id
       WHERE info = (
             SELECT MAX(info) FROM notes
             );

-- 15
SELECT nom, maxf, etudiant.classe FROM etudiant
       JOIN (
            SELECT classe AS maxclasse, MAX(francais) AS maxf FROM notes
                   JOIN etudiant ON notes.id_etudiant = etudiant.id
                   GROUP BY classe
       ) ON maxclasse = classe
       JOIN notes ON notes.id_etudiant = etudiant.id
       WHERE maxf = notes.francais;

-- 16
```

```sql
SELECT nom, prenom FROM etudiant ORDER BY nom ASC;


-- ========================================================
-- ccINP 2017
SELECT id_comptage, date, voie, q_exp, v_exp FROM comptages
       JOIN stations ON stations.id = comptages.id_station
       WHERE nom = 'M8B';

SELECT date, SUM(q_exp) FROM comptages_M8B
       GROUP BY date;

-- ========================================================
-- Mines-Ponts 2017
SELECT id_croisement_fin FROM voie
       WHERE id_croisement_debut = c;

SELECT longitude, latitude FROM croisement
       JOIN voie ON croisement.id = id_croisement_fin
       WHERE id_croisement_debut = c;

-- cette requête renvoie les identifiants des croisements
-- atteignables en utilisant deux voies à partir du
-- croisement ayant l'identifiant c,
-- avec donc un croisement intermédiaire.

-- ========================================================
-- X-ÉNS 2017
SELECT idensemble FROM membre
       JOIN points ON id = idpoint
       WHERE x = a AND y = b;

SELECT x, y FROM points
       JOIN membre AS m1 ON id = m1.idpoint
       JOIN membre AS m2 on id = m2.idpoint
       WHERE m1.idensemble = i
       AND m2.idensemble = j;

SELECT p.id FROM points AS p
       JOIN membre AS m ON m.idpoint = p.id
       JOIN membre AS m1 ON m1.idensemble = m.idensemble
       JOIN points AS p1 ON m1.idpoint = p1.id
       WHERE p1.x = a AND p1.y = b;

-- ========================================================
-- Centrale-Supélec 2017
SELECT ex_num FROM explo
       WHERE ex_deb IS NOT NULL
       AND ex_fin IS NULL;

SELECT pi_num, pi_x, pi_y FROM pi
       WHERE ex_num = 42;       -- par exemple, le 42

SELECT (MAX(pi_x)-MIN(pi_x))/1000 * (MAX(pi_y)-MIN(pi_y))/1000 FROM pi
       JOIN explo ON pi.ex_num = explo.ex_num
       WHERE ex_deb IS NOT NULL
       AND ex_fin IS NOT NULL   -- l'exploration est terminée
       GROUP BY pi.ex_num;

-- q4 : je ne comprends pas la question
-- Un entier est (?) stocké sur 64 bits, soit entre 0 et 2^64 - 1
-- Donc la surface maximale sera (2^64 - 1)^2=3.4e38 en mm2
-- soit 3.4e26 km2
-- En supposant bien-sûr que le calcul se fasse en basculant
-- automatiquement en flottants
-- sinon c'est 2^64 - 1=1.8e19 le plus grand nombre envisageable.

SELECT in_num, COUNT(*), SUM(it_dur) FROM intyp
       JOIN analy ON intyp.ty_num = analy.ty_num
       JOIN explo ON explo.ex_num = analy.ex_num
       WHERE ex_deb IS NOT NULL
       AND ex_fin IS NULL       -- l'exploration est en cours
       GROUP BY in_num;
```

```sql
-- ========================================================
-- Mines-Ponts 2016
-- Q1

-- Pour la table palu, iso peut servir de clé primaire
-- car deux lignes de la table ont un iso différent
```

```sql
-- Pour la table demographie, un seul attribut ne peut pas servir
-- de clé primaire (il peut y avoir deux fois la même population,
-- le même iso, la même période)
-- mais le couple (pays, periode) peut servir de clé primaire

-- Q2
SELECT * FROM palu
       WHERE annee = 2019
       AND deces >= 1000;

-- Q3
SELECT nom, cas/pop*100000 AS taux_d_incidence FROM palu
       JOIN demographie ON iso = pays AND annee = periode
       WHERE annee = 2011;

-- Q4
SELECT nom FROM palu
       WHERE annee = 2010
       AND cas = (
           SELECT MAX(cas) FROM palu
                  WHERE annee = 2019
                  AND cas != (
                      SELECT MAX(cas) FROM palu
                             WHERE annee = 2010
                  )
       );

-- ========================================================
-- Centrale-Supélec 2018
-- Q1
SELECT si_dim, count(*) FROM simulation
       GROUP BY si_dim;

-- Q2
SELECT si_num, count(*), AVG(re_vit) FROM rebond
       GROUP BY si.num;

-- Q3
SELECT re_dir, sum(2 * pa_m * re_vp) FROM rebond AS r
       JOIN particule AS p ON p.pa_num = r.pa_num
       WHERE si_num = n
       GROUP BY re_dir;

-- ========================================================
-- X-ÉNS 2016
-- Q1
SELECT id2 FROM liens WHERE id1 = x;

-- Q2
SELECT nom, prenom FROM individus
       JOIN liens ON id = id1
       WHERE id2 = x;

-- Q3
SELECT l1.id1 FROM liens AS l1
       JOIN liens AS L2 ON l1.id2 = l2.id1
       WHERE l2.id2 = x;
```

```python
# ========================================================
# X 2018
# ------------------------- Partie I -------------------------
# Question I.1
def SelectionConstante(table, indice, constante):
    T = []
    for e in table:
        if e[indice] == constante:
            T.append(e)
    return T

# Question I.2
# chaque tour de boucle s'exécute en temps constant,
# et il est fait $n$ tours de boucle, où $n$ est la taille de la
table
# La complexité est donc $O(n)$.

# Question I.3
def SelectionEgalite(table, indice1, indice2):
    T = []
    for e in table:
```

```
        if e[indice1] == e[indice2]:
            T.append(e)
    return T

# Question I.4
def ProjectionEnregistrement(enregistrement, listeIndices):
    l = []
    for i in listeIndice:
        l.append(enregistrement[i])
    return l

# Question I.5
def Projection(table, listeIndices):
    T = []
    for e in table:
        T.append(ProjectionEnregistrement(e, listeIndices))
    return T

# Question I.6
def ProduitCartesien(table1, table2):
    T = []
    for e1 in table1:
        for e2 in table2:
            T.append(e1+e2)
    return T

# Question I.7
# Définissons une première fonction qui prend en argument
# un enregistrement et un indice
# et qui renvoie l'enregistrement obtenu en supprimant l'attribut
# associé à l'indice
def Supprime(e, i):
    l = []
    for j in range(len(e)):
        if j != i:
            l.append(e[j])
    return l

def Jointure(table1, table2, indice1, indice2):
    T = []
    for e1 in table1:
        for e2 in table2:
            if e1[indice1] == e2[indice2]:
                T.append(e1 + Supprime(e2, indice2))
    return T

# Question I.8
# La fonction Supprime, appliquée à e2, consiste en une boucle
# de k_2 tours et chaque tour s'exécute en temps constant.
# Sa complexité est donc en O(k_2)
# La fonction Jointure consiste en une boucle de n_1 tours.
# Pour chacun de ces tours, il y a une boucle de n_2 tours.
# Pour chacun de ces tours, il y a
# zéro ou un appel à Supprime (O(k_2)),
# zéro ou une concaténation de listes (O(1) d'après l'énoncé)
# et zéro ou une utilisation de append (O(1)).
# La complexité est donc au maximum de O(n_1 × n_2 × k2)
# Pour être plus précis, si on note p le nombre d'enregistrements
# de la jointure, la complexité est de O(n_1 × n_2 × p).

# Question I.9
# On commence par définir une fonction qui teste l'égalité entre
# deux enregistrements qui sont supposés être de même longueur
def TesteEgalite(e1, e2):
    i = 0
    while i < len(e1) and e1[i] == e2[i]:
        # e1[0:i+1] == e2[0:i+1]
        i += 1
    # en sortie de boucle
    # soit i == n et e1[0:n] == e2[0:n]
    # soit i < n et e1[0:i] == e2[0:i] mais e1[i] != e2[i]
    return i == n

def SupprimerDoublons(table):
    n = len(table)
    T = []
    for i in range(n):
        j = i + 1
        while j < n and not(TesteEgalite(table[i], table[j])):
            # table[i] n'est égal à aucun des enregistrements
```

```
            # table[i+1], ..., table[j]
            j += 1
        # en sortie de boucle
        # soit j == n et table[i] n'est égal à aucun
        # des table[i+1]...table[n-1]
        # soit j < n et table[i] est égal à table[j]
        if j == n:
            T.append(table[i])
    return T

# Question I.10
# La fonction TesteEgalite a une complexité maximum de k.
# L'exécution de la fonction SupprimerDoublons consiste en une
boucle
# indexée par i au cours de laquelle on effectue au plus
# (n−i) tours de boucles avec pour chacun un appel à TesteEgalite
# Il y a donc au plus \sum_{i=0}^{n-1} \left( \sum_{j=i+1}^{n} k \right) = \sum_{i=0}^{n-1} k(n-i) = O(kn^2).

# ------------------------- Partie II -------------------------
# Question II.1
resultat = SelectionConstante(Trajet, 1, 'Rennes')

# Question II.2
resultat = ProduitCartesien(Trajet, Vehicule)

# Question II.3
r1 = ProduitCartesien(Trajet, Vehicule)
resultat = SelectionEgalité(r1, 3, 0)

# Question II.4
# Il y a un décalage d'indice pour la seconde table
# parce qu'on a supprimé l'attribut redondant
r1 = Jointure(Hotel, Chambre, 0, 1)
resultat = Projection(r1, [1,2,4,5])

# Question II.5
# Il s'agit en fait d'une jointure Hotel.Ville=Trajet.VilleA
r1 = Jointure(Hotel, Trajet, 2, 2)
# et d'une sélectionConstante
r2 = SelectionConstante(Ticket, 5, '50')
# On fait une jointure de ces deux tables
r3 = Jointure(r1,r2,3,1)
# et on projete pour avoir le résultat
resultat = Projection(r3, [0])

# Question II.6
# La sous-requête est celle de la question précédente.
# On la note r4
r1 = Jointure(Hotel, Trajet, 2, 2)
r2 = SelectionConstante(Ticket, 5, '50')
r3 = Jointure(r1,r2,3,1)
r4 = Projection(r3, [0])

r5 = SelectionConstante(Chambre, 3, '100')

# On termine par une jointure entre r4 et r5
resultat = Jointure(r4, r5, 0, 1)

# ------------------------- Partie III -------------------------
# Question III.1
def VerifieTrie(table, indice):
    n = len(table)
    i = 0
    while i < n-1 and table[i][indice] <= table[i+1][indice]:
        # table[0:i+2] est triée selon l'indice
        i += 1
    # en sortie de boucle
    # soit i = n-1 et table est triée selon l'indice
    # soit i < n-1 et table[0:i+1] est triée selon l'indice
    # mais table[i][indice] n'est pas <= table[i+1][indice]
    # donc la table n'est pas triée
    return i == n-1

# Question III.2
# on profite du fait que la table est triée
# pour faire une recherche dichotomique
def SelectionConstanteTrie(table, indice, constante):
    def aux(a,b):
```

```
        """Renvoie la table des enregistrements
        dont l'attribut indice vaut constante
        et dont le numéro dans la table est entre a et b"""
        if b < a :
            return []
        else:
            c = (a+b)//2
            if table[c][indice] < constante:
                # les enregistrements cherchés sont après c+1
                return aux(c+1,b)
            elif table[c][indice] > constante:
                # les enregistrements cherchés sont avant c-1
                return aux(a,c-1)
            else:
                # l'enregistrement c convient,
                # et on cherche avant et après
                return aux(a,c-1) + [table[c]] + aux(a,c+1)
    n = len(table)
    return aux(0, n-1)

# Question III.3
# On utilise deux indices i1 et i2
# pour parcourir simultanément les tables
# comme on le fait dans la fonction itérative de fusion du cours
def JointureTrie(table1, table2, indice1, indice2):
    n1, n2 = len(table1), len(table2)
    i1, i2 = 0, 0
    T = []
    while i1 < n1 and i2 < n2:
        if table1[i1][indice1] == table2[i2][indice2]:
            # les enregistrements sont à mettre dans T
            T.append(table1[i1] + Supprime(table2[i2], indice2))
        elif table1[i1][indice1] < table2[i2][indice2]:
            # l'enregistrement dans table1 est trop petit
            # pour correspondre à celui de
            # l'enregistrement de la table2
            i1 += 1
        else: # table1[i1][indice1] > table2[i2][indice2]:
            # l'enregistrement dans table2 est trop petit
            # pour correspondre à celui de
            # l'enregistrement de la table1
            i2 += 2
    return T

# Question III.4
# Le nombre de tours de boucles est majoré par n1 + n2.
# À chaque tour de boucle, il est fait des opérations en O(1)
# (gestion des indices, append, concaténation)
# et Supprime qui s'exécute en O(k2).
# On a donc une complexité en O((n1 + n2) × k2),
# ce qui est bien meilleur que O(n1 × n2 × k2)
# de la question I.8

# ------------------------- Utilisation de dictionnaire ------
# Question III.5
def CreerDictionnaire(table, indice):
    dico = {}
    n = len(table)
    for i in range(n):
        e = table[i]
        if e[indice] in dico:
            # la clé existe,
            # on ajoute i à la liste associée à la clé
            dico[e[indice]].append(i)
        else:
            # la clé n'existe pas encore, on la crée
            dico[e[indice]] = [i]
    return dico

# Question III.6
def SelectionConstanteDictionnaire(table, indice, constante, dico):
    if constante in dico:
        T = []
        for i in dico[constante]:
            T.append(table[i])
        return T
    else:
        return []

# Question III.7
```

```
# Comme toutes les opérations se font en temps constant,
# la complexité de SelectionConstanteDictionnaire
# est proportionnelle au nombre d'enregistrements
# correspondant à la valeur cherchée.
# Le gain est donc d'autant meilleur qu'on sélectionne
# sur une constante qui est dans peu d'enregistrement.
# En revanche, si la constante est présente
# dans tous les enregistrements, on n'y gagne rien.

# Question III.8
def JointureDictionnaire(table1, table2, indice1, indice2, dico2):
    T = []
    for e in table1:
        if e[indice1] in dico2:
            for i2 in dico2[e[indice1]]:
                T.append(e + Supprime(table2[i2], indice2))
    return T

# Question III.9
# Pour chaque enregistrement e de table1,
# on effectue une boucle d'au plus k_2 tours de boucle
# et au cours de chaque tour de boucle, des opérations en O(1)
# et O(k_2) pour Supprime
# Soit au total O(n_1 × k_2^2)

# Question III.10
# Si on inverse le rôle des deux tables,
# la complexité devient O(n_2 × k_1^2).
# Il peut donc être intéressant d'indexer la table
# contenant le plus d'enregistrements
```