Université Sorbonne Paris Nord
École d'ingénieur Sup Galilée

TP2 : Requêtes, dépendances fonctionnelles et normalisation

Rapport de TP

Réalisé par :

Célian WIRTZ (celian.wirtz@edu.univ-paris13.fr)

12100311

Date • 27/04/25

Année universitaire : 2024/2025

Responsable

M. Youcef

Exercice n° 1

```
-- 1
SELECT dept_name FROM department WHERE budget = (SELECT
MAX(budget) FROM department);

-- 2
SELECT name, salary FROM teacher WHERE salary > (SELECT AVG(salary)
FROM teacher);

-- 3
SELECT t.name AS prof, s.name AS eleve, COUNT(*) AS nb_cours FROM
teacher t
JOIN teaches te ON t.id = te.id JOIN takes ta
ON (ta.course_id, ta.sec_id, ta.semester, ta.year) = (te.course_id,
te.sec_id, te.semester, te.year)
JOIN student s ON ta.ID = s.ID GROUP BY t.name, s.name HAVING
COUNT(*) >= 2;

-- 4
SELECT t.name AS prof, s.name AS eleve, sub.nb_cours FROM (SELECT
te.id, ta.id AS student_id,
COUNT(*) AS nb_cours FROM teaches te JOIN takes ta
ON (ta.course_id, ta.sec_id, ta.semester, ta.year) = (te.course_id,
te.sec_id, te.semester, te.year)
GROUP BY te.id, ta.id) AS sub JOIN teacher t ON t.id = sub.id JOIN student s
ON s.id = sub.student_id
WHERE sub.nb_cours >= 2;
```

```
-- 5
SELECT s.ID, s.name FROM student s
WHERE NOT EXISTS (SELECT 1 FROM takes t WHERE t.ID = s.ID AND t.year
< 2010);

-- 6
SELECT id, name FROM teacher WHERE name LIKE 'E%';

-- 7
SELECT salary, name FROM teacher WHERE salary =
(SELECT DISTINCT salary FROM teacher ORDER BY salary DESC LIMIT 1
OFFSET 3);

-- 8
SELECT name, salary FROM (SELECT name, salary FROM teacher ORDER BY
salary ASC LIMIT 3) AS ptit_salaire
ORDER BY salary DESC;

-- 9
SELECT name FROM student
WHERE id IN (SELECT DISTINCT id FROM takes WHERE semester = 'Fall'
AND year = 2009);

-- 10
SELECT name FROM student
WHERE id = SOME (SELECT id FROM takes WHERE semester = 'Fall' AND
year = 2009);
```

```
-- 11
SELECT DISTINCT s.name FROM student s
NATURAL JOIN takes t WHERE t.semester = 'Fall' AND t.year = 2009;
-- Dans la version de sql que j'utilise je dois utiliser NATURAL JOIN pas
NATURAL INNER JOIN
-- (donc si ça marche pas pour vous faut juste rajouter INNER)

-- 12
SELECT s.name FROM student s
WHERE EXISTS (SELECT 1 FROM takes t WHERE t.ID = s.ID AND t.semester =
'Fall' AND t.year = 2009);

-- 13
SELECT DISTINCT s1.name AS E1, s2.name AS E2 FROM takes t1
JOIN takes t2 ON t1.course_id = t2.course_id
AND t1.sec_id = t2.sec_id AND t1.semester = t2.semester AND t1.year =
t2.year
AND t1.ID < t2.ID -- Pas de paire avec lui même ni de doublons (les deux en
même temps)
JOIN student s1 ON t1.ID = s1.ID JOIN student s2 ON t2.ID = s2.ID;

-- 14
SELECT t.name AS prof, COUNT(ta.ID) AS nb_eleve FROM teacher t
JOIN teaches te ON t.ID = te.ID LEFT JOIN takes ta
ON (te.course_id, te.sec_id, te.semester, te.year) = (ta.course_id,
ta.sec_id, ta.semester, ta.year)
```

```
GROUP BY t.ID, t.name ORDER BY nb_eleve DESC;

--15
SELECT t.name AS prof, COUNT(ta.ID) AS nb_eleve FROM teacher t
LEFT JOIN teaches te ON t.ID = te.ID LEFT JOIN takes ta
ON (te.course_id, te.sec_id, te.semester, te.year) = (ta.course_id,
ta.sec_id, ta.semester, ta.year)
GROUP BY t.ID, t.name ORDER BY nb_eleve DESC;

-- 17
SELECT t.name AS prof, s.name AS eleve, COUNT(*) AS nb_cours FROM
teacher t
JOIN teaches te ON t.ID = te.ID
JOIN takes ta ON (te.course_id, te.sec_id, te.semester, te.year) =
(ta.course_id, ta.sec_id, ta.semester, ta.year)
JOIN student s ON ta.ID = s.ID GROUP BY t.ID, s.ID ORDER BY prof;

-- 18
SELECT t.name AS prof, s.name AS eleve, COUNT(*) AS nb_cours FROM
teacher t
JOIN teaches te ON t.ID = te.ID
JOIN takes ta ON (te.course_id, te.sec_id, te.semester, te.year) =
(ta.course_id, ta.sec_id, ta.semester, ta.year)
JOIN student s ON ta.ID = s.ID GROUP BY t.ID, s.ID HAVING COUNT(*) >= 2
ORDER BY prof;
```

Exercice n° 2

1. R(A, B, C) et F = {A −→ B; B −→ C}

Clé primaire : A

Cette relation est en 2NF (pas en 3NF car il y a une transitivité)

Ainsi on décompose en 2 relations :

R1(B, C) et F1 = {B −→ C} et R2(A, B) et F2 = {A −→ B}

Ces deux relations sont bien en forme normale (B est super-clé pour R1 et A pour R2)

2. R(A, B, C) et F = {A −→ C; A −→ B}

Cette relation est déjà en BCNF (pas besoin de normaliser)

3. R(A, B, C) et F = {A, B −→ C; C −→ B}

Cette relation est en 3NF cependant, C n'étant pas une super-clé, la relation n'est pas en BCNF.

Ainsi on décompose en 2 relations :

R1(B, C) et F1 = {C −→ B} et R2(A, B) et F2 = {}

Exercice n° 3

Question 1 :

R(A, B, C, D, E) et F = {A −→ B,C; C, D −→ E; B −→ D; E −→ A}

Réflexivité : A → A / B → B / C → C / D → D / E → E / A,B → A,B ...

Transitivité : A → D,C / A → E,C / C,D → A / C,D → B / C,D → D / C,D → E / B → E / B→ A / E → B / E → D

Augmentation : A,B → B,C / A,C → B,C / A,D → B,C / A,E → B,C / A → B,C,D / ….

Question 2 :

R(A, B, C, D, E, F) et F = {A −→ B,C,D; B,C −→ D,E; B −→ D; D −→ A}.

(a) Calculer la fermeture de l'attribut B et de l'ensemble {A, B}

B → B (réflexivité) et B→ D donc B → B,D or D → A donc B → A,B,D or A → B,C,D donc B → A,B,C,D or B,C → D,E donc B → A,B,C,D,E ainsi la fermeture de B est {A,B,C,D,E}.

B appartient à {A,B} donc la fermeture de B est incluse dans celle de {A,B} or la fermeture de B est déjà remplie au maximum (sans F mais F n'étant même pas dans une DF elle ne pourra pas apparaître dans la fermeture d'un ensemble ne contenant pas F) donc la fermeture de {A,B} est {A,B,C,D,E}.

On sait que : A −→ B,C,D; donc A → B or B → A,B,C,D,E ainsi A,F → B,F → A,B,C,D,E,F donc A,F est super clé de cette relation.

A n'étant même pas une super clé, cette relation n'est pas en BCNF.

On décompose R en R1(A, B, C, D) et R2(A, E, F) avec F1 = {A → B,C,D ; B → D ; D → A} et F2 = {}.

R2 est en BCNF.

R1 n'est pas en BCNF (B n'est pas super-clé)

On décompose R1 en R3(B, D) et R4(A, C) avec F3 = {B → D} et F4 = {A → C }.

R3 et R4 étant en BCNF, on a la décomposition finale suivante :

R2(A, E, F) et F2 = {}

R3(B, D) et F3 = {B → D}

R4(A, C) et F4 = {A → C }

Question 3 :

(On reprend ceux de la question précédente ???)

R1(A, B, C) et R2(A, D, E)

Attribut commun : A

A est clé dans les deux relations (donc dans au moins une) : A → B,C et A → D,E

Donc cette décomposition est sans perte d'information.

R1(A, B, C) R2(C, D, E)

Attribut commun : C

C n'est clé dans aucune des deux relations, donc cette décomposition est avec perte d'information.

Exercice n° 4

```python
import itertools

myrelations = [{"A", "B", "C", "G", "H", "I"}, {"X", "Y"}]
mydependencies = [ [ {"A"}, {"B"} ], [ {"A"}, {"C"} ], [ {"C", "G"}, {"H"} ],
[ {"C", "G"}, {"I"} ], [ {"B"}, {"H"} ], [{"H"}, {"R"}]]


# Question 1 (donné par le prof)
def printDependencies(dep_liste):
    print("My Dependencies :")
    for alpha , beta in dep_liste:
        print ("\t", alpha , " --> ", beta)
```

```python
# Question 2 (donné par le prof)
def printRelations(rel_liste):
    print("My Relations :")
    for R in rel_liste:
        print ("\t", R)


# Question 3 (donné par le prof)
def powerSet(inputset):
    _result = []
    for r in range(1, len (inputset) + 1):
        _result = _result + list(map(set, itertools.combinations(inputset, r)))
    return _result


# Question 4
def fermeture(dep_liste, attribut_liste):
    _fermeture = set(attribut_liste)
    _fermeture2 = set()

    while(_fermeture != _fermeture2):
        _fermeture2 = _fermeture.copy()
        _powset = powerSet(_fermeture)

        for i, j in dep_liste:
            if i in _powset:
                _fermeture.update(j)
    return _fermeture


# Question 5
# Bon avec l'exemple il y en a beaucoup mais c'était à prévoir car il y a énormément de combinaisons possibles.
def cloture(dep_liste):
    R = set ()
    for a,b in dep_liste: R.update(a | b)

    _cloture = []

    for i in powerSet(R) :
        for j in powerSet(fermeture(dep_liste,i)) :
            _cloture.append([i, j])
    return _cloture


# Question 6
# On vérifie que les a implique b à l'aide de la fermeture
def implique(dep_liste, a, b):
    return b.issubset(fermeture(dep_liste, a))

# Question 7
# On vérifie que chaque élément de la relation soit bien un élément de la fermeture de l'attribut (et donc que l'attribut soit super-clé)
def supercle(dep_liste, attribut, relation):
    return relation.issubset(fermeture(dep_liste, attribut))
```

```python
# Question 8
# Suffit de vérifier que c'est une super clé et que quand on retire un élément ça ne l'est plus
def candidate(dep_liste, attribut, relation):
    if not(supercle(dep_liste, attribut, relation)) :
        return False
    else :
        for i in attribut:
            temp = set(attribut)
            temp.discard(i)
            if supercle(dep_liste, temp, relation):
                return False
    return True


# Question 9
# Bon bah on teste toutes les possibilités... Bon par contre il y a peut-être mieux d'un point de vue complexité
def liste_candidate(dep_liste, relation):
    L = []
    for i in powerSet(relation):
        if candidate(dep_liste, i, relation) :
            L.append(i)
    return L


# Question 10
# Bon bah on teste toutes les possibilités encore :) ...
def liste_supercle(dep_liste, relation):
    L = []
    for i in powerSet(relation):
        if supercle(dep_liste, i, relation) :
            L.append(i)
    return L


import random
# Question 11
# Euh bah on en pioche une dans la liste c'est tout...
def random_candidate(dep_liste, relation):
    return random.choice(liste_candidate(dep_liste, relation))


# Question 12
# On vérifie que chaque élément de gauche des DF est une super clé.
def BCNF(dep_liste, relation):
    for a, b in dep_liste:
        if not supercle(dep_liste, a, relation) :
            return False
    return True
```

```python
if (__name__ == "__main__"):

    _bool = ""

    #printDependencies(mydependencies)
    #printRelations(myrelations)

    #choisir le result que vous voulez (chacun correspond à une question)
    #_result = powerSet({'A', 'B', 'C'})
    #_result = fermeture(mydependencies, {'B', 'G'})
    #_result = cloture(mydependencies)

    #_bool = implique(mydependencies, {'A'}, {'H'})
    #_bool = supercle(mydependencies, {'A', 'G'}, myrelations[0])

    #_bool = candidate(mydependencies, {'A'}, myrelations[0])
    #_bool = candidate(mydependencies, {'A', 'G'}, myrelations[0])
    #_bool = candidate(mydependencies, {'A', 'C', 'G'}, myrelations[0])

    #_result = liste_candidate(mydependencies, myrelations[0])
    #_result = liste_supercle(mydependencies, myrelations[0])
    #_result = random_candidate(mydependencies, myrelations[0])

    #_bool = BCNF(mydependencies, myrelations[0])

    if '_result' in locals():
       pass
    else:
        _result = []

    for i in _result:
        print("\t", i)

    print("\t", _bool)

    exit(0)
```