**Seyfeddine Jouini**
**N° Étudiant : 12413361**

# Compte rendu TP 2

## Exercice 1 :

**1.**
```
select dept_name, budget from
        department     where budget in (select max(budget) from department)
```

**2.**
```
select teacher_name, salary from
        teacher          where salary >(select avg(salary) from teacher)
```

**3.**
```
select teacher.teacher_name, student.student_name, count(*) as nombre_cours  from
        teacher, student, takes,teaches where
        takes.takes_id = student.student_id and teacher.teacher_id= teaches.teaches_id and
        takes.course_id =teaches.course_id and takes.sec_id = teaches.sec_id and
        takes.semester = teaches.semester and takes.takes_year = teaches.teaches_year
GROUP BY teacher.teacher_name , student.student_name HAVING count (*) >= 2;
```

**4.**
```
select S.teacher_name, S.student_name, S.nombre_cours
from (
   select teacher.teacher_name as teacher_name,
         student.student_name as student_name,
         count(*) as nombre_cours
   from teacher
   join teaches on teacher.teacher_id = teaches.teaches_id
   join takes on takes.course_id = teaches.course_id
           and takes.sec_id = teaches.sec_id
           and takes.semester = teaches.semester
           and takes.takes_year = teaches.teaches_year
   join student on takes.takes_id = student.student_id
   group by teacher.teacher_name, student.student_name
) S
where S.nombre_cours >= 2;
```

**5.**
```
select student.student_id, student.student_name
from student
where student.student_id not in (
        select takes.takes_id
        from takes
        where takes.takes_year < 2009
);
```

**6.**
```
select teacher_name from teacher where teacher_name LIKE 'E%' ;
```

**7.**
```
select t1.teacher_name
from teacher t1
where 3 = (
        select count(distinct t2.salary)
        from teacher t2
        where t2.salary > t1.salary
);
```

**8.**
```
select t1.teacher_name, t1.salary
from teacher t1
where 2 >= (
        select count(distinct t2.salary)
        from teacher t2
        where t2.salary < t1.salary
)
order by t1.salary desc;
```

**9.**
```
select s.student_name
from student s
where (s.student_id) in (
        select takes.takes_id
        from takes
        where takes.semester = 'Fall'
        and takes.takes_year = 2009
);
```

**10.**
```
select s.student_name
from student s
where ( 'Fall', 2009 ) = some (
        select takes.semester, takes.takes_year
        from takes
        where takes.takes_id = s.student_id
);
```

**11.**
```
select s.student_name
from student s
natural inner join takes t
where t.semester = 'Fall'
  and t.takes_year = 2009;
```

**12.**
```
SELECT student_name
FROM student
WHERE EXISTS ( SELECT *
```

```
FROM takes
WHERE takes.takes_id = student.student_id AND semester = 'Fall'
AND takes_year = 2009) ;
```
**13.**
```
select a.student_name, b.student_name
from (student natural inner join takes) as a,
        (student natural inner join takes) as b
where a.course_id = b.course_id
  and a.sec_id = b.sec_id
  and a.semester = b.semester
  and a.takes_year = b.takes_year
  and a.student_id <> b.student_id
  and a.student_name < b.student_name
group by a.student_id, b.student_id
having count(*) >= 1;
```
**14.**
```
SELECT teacher.teacher_name , count (*)
from takes INNER JOIN teaches ON takes.course_id = teaches.course_id
                AND takes.sec_id = teaches.sec_id
                AND takes.semester = teaches.semester
                AND takes.takes_year = teaches.teaches_year
INNER JOIN teacher ON teaches.teaches_id = teacher.teacher_id
GROUP BY teacher.teacher_name , teacher.teacher_id ORDER BY count (*) DESC ;
```
**15.**
```
SELECT teacher.teacher_name, count(teaches.course_id)
FROM takes
INNER JOIN teaches ON takes.course_id = teaches.course_id
                AND takes.sec_id = teaches.sec_id
                AND takes.semester = teaches.semester
                AND takes.takes_year = teaches.teaches_year
RIGHT OUTER JOIN teacher ON teaches.teaches_id = teacher.teacher_id
GROUP BY teacher.teacher_name, teacher.teacher_id
ORDER BY count(teaches.course_id) DESC;
```
**16**.
```
WITH mytakes (id, course_id, sec_id, semester, year, grade) AS (
        SELECT takes_id, course_id, sec_id, semester, takes_year, grade
        FROM takes
        WHERE grade = 'A'
)
SELECT teacher.teacher_name, count(mytakes.course_id)
FROM mytakes
INNER JOIN teaches ON mytakes.course_id = teaches.course_id
                AND mytakes.sec_id = teaches.sec_id
                AND mytakes.semester = teaches.semester
                AND mytakes.year = teaches.teaches_year
RIGHT OUTER JOIN teacher ON teaches.teaches_id = teacher.teacher_id
GROUP BY teacher.teacher_name, teacher.teacher_id
ORDER BY count(mytakes.course_id) DESC;
```

**17**. SELECT teacher.teacher_name, student.student_name, count(*)
FROM teacher
INNER JOIN teaches ON teacher.teacher_id = teaches.teaches_id
INNER JOIN takes ON teaches.course_id = takes.course_id
AND teaches.sec_id = takes.sec_id
AND teaches.semester = takes.semester
AND teaches.teaches_year = takes.takes_year
INNER JOIN student ON takes.id = student.student_id
GROUP BY teacher.teacher_name, student.student_name;

**18.** SELECT teacher.teacher_name, student.student_name, count(*)
FROM teacher
INNER JOIN teaches ON teacher.teacher_id = teaches.teaches_id
INNER JOIN takes ON teaches.course_id = takes.course_id
AND teaches.sec_id = takes.sec_id
AND teaches.semester = takes.semester
AND teaches.teaches_year = takes.takes_year
INNER JOIN student ON takes.id = student.student_id
GROUP BY teacher.teacher_name, student.student_name;

**Exercice 3 :**
**1-**A → A (Réflexivité)
B → B (Réflexivité)
C → C (Réflexivité)
D → D (Réflexivité)
E → E (Réflexivité)
A → B (Décomposition de A → BC)
A → C (Décomposition de A → BC)
A → D (Transitivité de A → B et B → D)
E → A (Donnée)
E → B (Transitivité de E → A et A → B)
E → C (Transitivité de E → A et A → C)
CD → E (Donnée)
CD → A (Transitivité de CD → E et E → A)
B → D (Donnée)
A → BC (Donnée)
E → BC (Transitivité de E → A et A → BC)
**2-**
**a-** $B^+$ = {B, D, A, C, E}
$(AB)^+$ = {A, B, C, D, E}
**b-** $(AF)^+$ = {A, B, C, D,E, F} => donc (AF) est une super clé.

**3/a-** Décomposition en R1(A,B,C) et R2(A,D,E) :
Les ensembles d'attributs sont :

- Attributs(R1) = {A, B, C}
- Attributs(R2) = {A, D, E}

L'intersection des attributs est :

- Attributs(R1) ∩ Attributs(R2) = {A}

Si A est une super clé, alors la décomposition est sans perte.
**3/b-**
Les ensembles d'attributs sont :

- Attributs(R1) = {A, B, C}
- Attributs(R2) = {C, D, E}

L'intersection des attributs est :

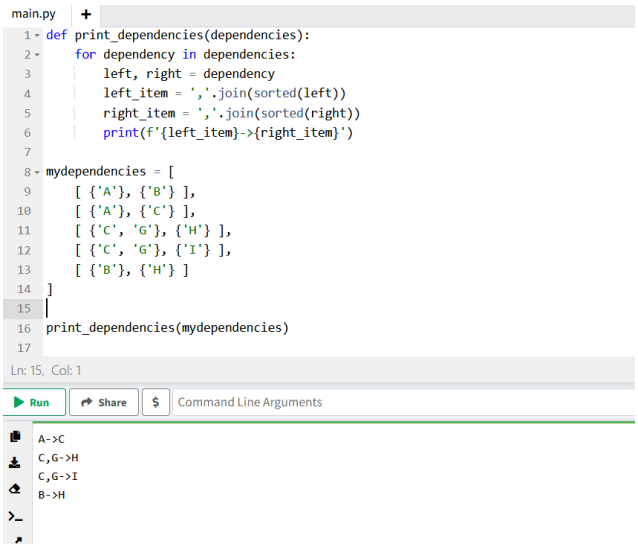
- Attributs(R1) ∩ Attributs(R2) = {C}

Si C est une super clé, alors la décomposition est sans perte.

**Exercice 4 :**

**1-**def print_dependencies(dependencies):
    for dependency in dependencies:
        left, right = dependency
        left_item = ','.join(sorted(left))
        right_item = ','.join(sorted(right))
        print(f'{left_item}->{right_item}')

Execution :




2- def print_relations(relations):

for relation in relations:
if isinstance(relation, set):
print(f'R({", ".join(sorted(relation))})')

```
def print_relations(relations):
    for relation in relations:
        if isinstance(relation, set):
            print(f'R({", ".join(sorted(relation))})')

myrelations = [
    {'A', 'B', 'C', 'G', 'H', 'I'},
    {'X', 'Y'}
]

print_relations(myrelations)
```

27, Col: 1

Run | Share | $ | Command Line Arguments

```
R(A, B, C, G, H, I)
R(X, Y)


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

3- def powerSett(inputset):
subset_list = []
for i in range(len(inputset) + 1):
for subset in itertools.combinations(inputset, i):
subset_list.append(set(subset))
return subset_list

```
30
31 def power_set(inputset):
32     subset_list = []
33     for i in range(len(inputset) + 1):
34         for subset in itertools.combinations(inputset, i):
35             subset_list.append(set(subset))
36     return subset_list
37
38 example_set = {'A', 'B', 'C'}
39 print("Power Set:", power_set(example_set))
40
```

Ln: 32, Col: 21

Run | Share | $ | Command Line Arguments

```
Power Set: [set(), {'C'}, {'A'}, {'B'}, {'C', 'A'}, {'C', 'B'}, {'A', 'B'}, {'C', 'B', 'A'}]


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

4- def closure(attributes, dependencies):
closure_set = set(attributes)
changed = True
while changed:

changed = False
for left, right in dependencies:
if left.issubset(closure_set) and not right.issubset(closure_set):
closure_set.update(right)
changed = True
return closure_set

```
def closure(attributes, dependencies):
    closure_set = set(attributes)
    changed = True
    while changed:
        changed = False
        for left, right in dependencies:
            if left.issubset(closure_set) and not right.issubset(closure_set):
                closure_set.update(right)
                changed = True
    return closure_set

attributes = {'A'}
print(attributes, "+ = ", closure(attributes, mydependencies))
```

Ln: 53, Col: 24

Run | Share | $ | Command Line Arguments

```
{'A'} + =  {'H', 'C', 'A', 'B'}
```

6. def determinates_functionally(F, alpha, beta):
closure_alfa = closure(alpha, F)
return beta.issubset(closure_alfa)

```
def determinates_functionally(F, alpha, beta):
    closure_alfa = closure(alpha, F)
    return beta.issubset(closure_alfa)

alpha_att = {'A'}
beta_att = {'B','C','Z'}
print(determinates_functionally(mydependencies,alpha_att, beta_att))
```

Ln: 62, Col: 23

Run | Share | $ | Command Line Arguments

```
False


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

7.def IsSuperKey(dependencies, relation, K):
return relation.issubset(closure(K,mydependencies))

```
relation= {'A', 'B', 'C','H'}
K={'A'}
def IsSuperKey(dependencies, relation, K):
    return relation.issubset(closure(K,mydependencies))
print(K , "is a superkey of ", relation, "= ", IsSuperKey(mydependencies,relation,K))
```

65, Col: 29

Run | Share | $ | Command Line Arguments

```
{'A'} is a superkey of  {'C', 'B', 'A', 'H'} =  True


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

8. def IsCandidateKey (F, R, K):
    if not IsSuperKey(F,R,K) :
    return False
    if not IsSuperKey (F , R , K ) : return False
    for A in K :
    K_aux = set (K)
    K_aux.discard (A)
    if IsSuperKey (F , R , K_aux ) : return False
    return True

```
relation= {'A', 'B', 'C','H'}
K={'A'}
def IsCandidateKey (F, R, K):
    if not IsSuperKey(F,R,K) :
        return False
    if not IsSuperKey (F , R , K ) : return False
    for A in K :
        K_aux = set (K)
        K_aux.discard (A)
        if IsSuperKey (F , R , K_aux ) : return False
    return True
print(IsCandidateKey(mydependencies, relation, K))
```

n: 80, Col: 37

Run | Share | $ | Command Line Arguments

```
True
```

```
relation= {'A', 'B', 'C','H'}
K={'A','B'}
def IsCandidateKey (F, R, K):
    if not IsSuperKey(F,R,K) :
        return False
    if not IsSuperKey (F , R , K ) : return False
    for A in K :
        K_aux = set (K)
        K_aux.discard (A)
        if IsSuperKey (F , R , K_aux ) : return False
    return True
print(IsCandidateKey(mydependencies, relation, K))
```

n: 72, Col: 10

Run | Share | $ | Command Line Arguments

```
False
```

9.def compute_all_candidate_keys(F, R):
    result = []
    for K in power_set(R):
    if is_candidate_key(F, R, K):
    result.append(K)
    return result

```python
def compute_all_candidate_keys(F, R):
    result = []
    for K in power_set(R):
        if is_candidate_key(F, R, K):
            result.append(K)
    return result

R = {'A', 'B', 'C', 'D', 'E'}
F = [
    ( {'A', 'B'}, {'C'} ),
    ( {'C'}, {'D', 'E'} )
]

candidate_keys = compute_all_candidate_keys(F, R)
print("Clés candidates:", [sorted(k) for k in candidate_keys])
```

55, Col: 29

Run | Share | $ | Command Line Arguments

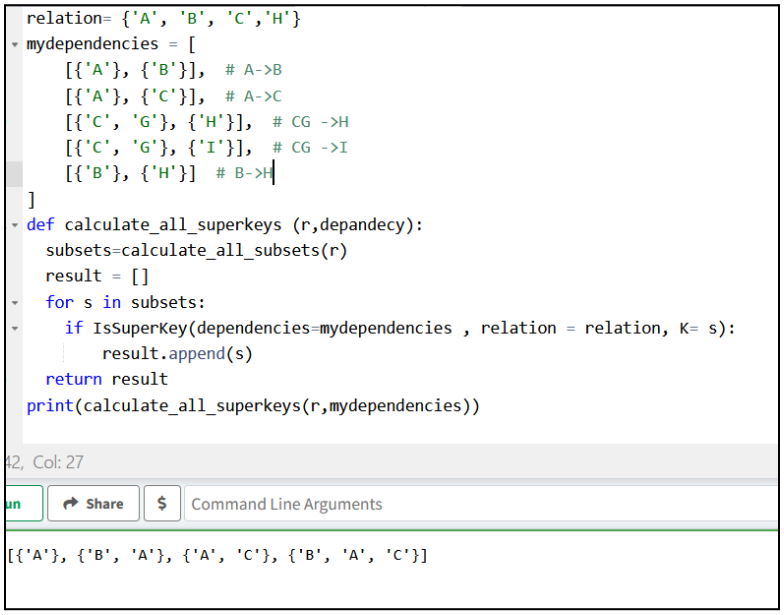
```
Clés candidates: [['A', 'B']]
```

10. Cet opération se fait en deux étapes:
    1. Calculer toutes les sous-séquences de la relation :

```
def calculate_all_subsets(r):
    subsets = [set()]
    for element in r:
        new_subsets = []
        for subset in subsets:
        new_subsets.append(subset.union({element}))
        subsets.extend(new_subsets)
    return subsets
```

    2. On va prendre les sous-séquences qui sont des super clés :

```
def calculate_all_superkeys (r,depandecy):
    subsets=calculate_all_subsets(r)
    result = []
    for s in subsets:
        if IsSuperKey(dependencies=mydependencies , relation = relation, K= s):
        result.append(s)
    return result
```



```
relation= {'A', 'B', 'C','H'}
mydependencies = [
    [{'A'}, {'B'}],  # A->B
    [{'A'}, {'C'}],  # A->C
    [{'C', 'G'}, {'H'}],  # CG ->H
    [{'C', 'G'}, {'I'}],  # CG ->I
    [{'B'}, {'H'}]  # B->H
]
def calculate_all_superkeys (r,depandecy):
  subsets=calculate_all_subsets(r)
  result = []
  for s in subsets:
    if IsSuperKey(dependencies=mydependencies , relation = relation, K= s):
        result.append(s)
  return result
print(calculate_all_superkeys(r,mydependencies))
```

42, Col: 27

un | Share | $ | Command Line Arguments

```
[{'A'}, {'B', 'A'}, {'A', 'C'}, {'B', 'A', 'C'}]
```

11.def find_one_candidate_key(R, F):
all_subsets = power_set(R)
all_subsets.sort(key=len)

for subset in all_subsets:
if is_candidate_key(subset, R, F):
return subset

return None

```python
def find_one_candidate_key(R, F):
    all_subsets = power_set(R)
    all_subsets.sort(key=len)

    for subset in all_subsets:
        if is_candidate_key(subset, R, F):
            return subset

    return None
```

RQ: Au début, on a fait un tri pour au final avoir la clé la plus courte.

Exemple :

```python
if __name__ == "__main__":
    R = {'A', 'B', 'C', 'D', 'E','H'}
    F = [
        ({'A', 'B'}, {'C'}),
        ({'C'}, {'D', 'E'}),
        ({'H'}, {'E','D'})
    ]

    key = find_one_candidate_key(R, F)
    if key:
        print("Une clé candidate trouvée :", sorted(key))
    else:
        print("Aucune clé candidate n'a été trouvée.")
```

2, Col: 30

un | Share | $ | Command Line Arguments

```
Une clé candidate trouvée : ['A', 'B', 'H']
```

12. def est_en_BCNF(relation, dependencies):
    for left, right in dependencies:
    if not left:
    continue
    if not relation.issubset(closure(left, dependencies)):
    return False
    return True

```
131
132 myrelations = [
133     {'A', 'B', 'C', 'G', 'H', 'I'},
134     {'X', 'Y'}
135 ]
136
137 for r in myrelations:
138     if est_en_BCNF(r, mydependencies):
139         print(f"La relation R({', '.join(sorted(r))}) est en BCNF.")
140     else:
141         print(f"La relation R({', '.join(sorted(r))}) n'est pas en BCNF.")
```

Ln: 147, Col: 2

Run | Share | $ | Command Line Arguments

```
La relation R(A, B, C, G, H, I) n'est pas en BCNF.
```

13.def est_en_BCNF_schema(schema_relations, dependances_fonctionnelles):
    def est_en_BCNF_relation(relation, dependencies_relation):

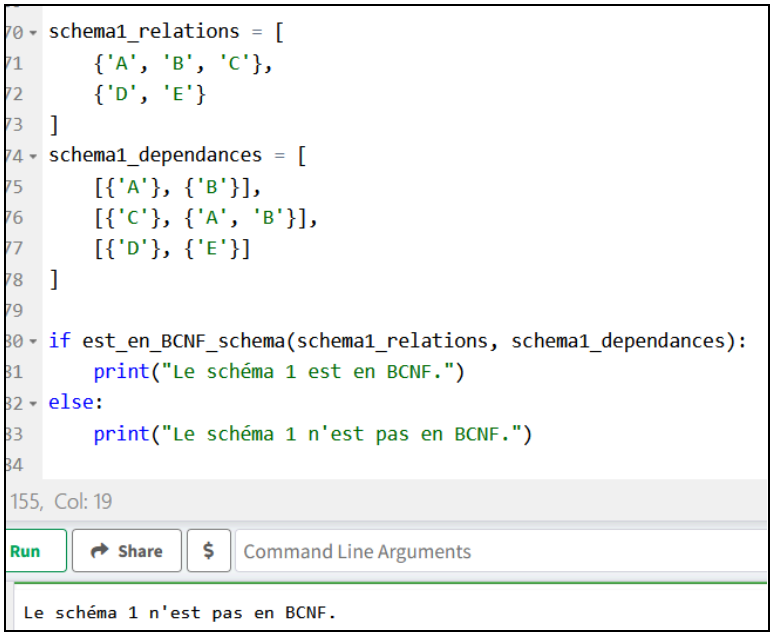
    for left, right in dependencies_relation:
    if not left:
        continue
    if not relation.issubset(closure(left, dependencies_relation)):
        return False
    return True

    for relation in schema_relations:

    dependances_relation = []
    for dep_left, dep_right in dependances_fonctionnelles:
    if dep_left.issubset(relation) and dep_right.issubset(relation):
        dependances_relation.append([dep_left, dep_right])

    if not est_en_BCNF_relation(relation, dependances_relation):
    return False

    return True




14.def decomposition_BCNF(relations, dependances_fonctionnelles):
result = list(relations)  # Commencer avec les relations initiales
i = 0
while i < len(result):
R = result[i]
Fd_R = []  # Dépendances fonctionnelles applicables à R
for left, right in dependances_fonctionnelles:
if left.issubset(R) and right.issubset(R):
Fd_R.append([left, right])

if not est_en_BCNF_relation(R, Fd_R):
# Trouver une dépendance fonctionnelle X -> Y dans Fd_R qui viole BCNF
violating_fd = None
for left, right in Fd_R:
if not left:
continue
if not R.issubset(fermeture(left, Fd_R)):
violating_fd = [left, right]
break

if violating_fd:
X, Y = violating_fd
R1 = X.union(Y)
R2 = R.difference(Y)

# Ajouter les nouvelles relations à la liste et supprimer l'ancienne
if R1 not in result:
result.append(R1)

```
            if R2 and R2 not in result:  # Ajouter R2 seulement s'il n'est pas vide
                result.append(R2)
                result.pop(i)
                i = 0  # Recommencer la vérification depuis le début
        else:
            i += 1  # Aucune violation trouvée (cas improbable si la condition BCNF est fausse)
        else:
        i += 1

        return result
```