TP2 – BDA

Baba SOW **ING – INFO2**

2024/2025

Exercic1 :

1. Afficher le nom du département qui a le budget le plus élevé.

```
1  SELECT dept_name
2  FROM department
3  WHERE budget = (SELECT MAX(budget) FROM department);
4
5
```

Results | Explain | Describe | Saved SQL | History

DEPT_NAME
Finance

1 rows returned in 0.02 seconds Download

2. Afficher les salaires et les noms des enseignants qui gagnent plus que le salaire moyen.

```
1  SELECT teacher_name, salary
2  FROM teacher
3  WHERE salary > (SELECT AVG(salary) FROM teacher);
4
```

Results | Explain | Describe | Saved SQL | History

TEACHER_NAME	SALARY
Wu	90000
Einstein	95000
Gold	87000
Katz	75000
Singh	80000
Brandt	92000
Kim	80000

7 rows returned in 0.01 seconds Download

3. Pour chaque enseignant, afficher tous les étudiants qui ont suivi plus de deux cours dispensés par cet enseignant ainsi que le nombre total de cours suivis par chaque étudiant, en utilisant la clause HAVING.

```sql
SELECT teacher.teacher_name, student.student_name, COUNT(*) AS total_cours
FROM teacher
JOIN teaches ON teacher.teacher_id = teaches.teacher_id
JOIN takes ON teaches.course_id = takes.course_id
          AND teaches.sec_id = takes.sec_id
          AND teaches.semester = takes.semester
          AND teaches.semester_year = takes.semester_year
JOIN student ON takes.student_id = student.student_id
GROUP BY teacher.teacher_name, student.student_name
HAVING COUNT(*) > 2;
```

Results | Explain | Describe | Saved SQL | History

TEACHER_NAME	STUDENT_NAME	TOTAL_COURS
Srinivasan	Shankar	3

1 rows returned in 0.06 seconds Download

4. Pour chaque enseignant, afficher tous les étudiants qui ont suivi plus de deux cours dispensés par cet enseignant ainsi que le nombre total de cours suivis par chaque étudiant, sans utiliser la clause HAVING.

```sql
SELECT T.teacher_name, T.student_name, T.total_cours
FROM (
    SELECT teacher.teacher_name, student.student_name, COUNT(*) AS total_cours
    FROM teacher
    JOIN teaches ON teacher.teacher_id = teaches.teacher_id
    JOIN takes ON teaches.course_id = takes.course_id
              AND teaches.sec_id = takes.sec_id
              AND teaches.semester = takes.semester
              AND teaches.semester_year = takes.semester_year
    JOIN student ON takes.student_id = student.student_id
    GROUP BY teacher.teacher_name, student.student_name
) T
WHERE T.total_cours > 2
ORDER BY T.teacher_name;
```

Results | Explain | Describe | Saved SQL | History

TEACHER_NAME	STUDENT_NAME	TOTAL_COURS
Srinivasan	Shankar	3

1 rows returned in 0.07 seconds Download

5. Afficher les identifiants et les noms des étudiants qui n'ont pas suivi de cours avant 2010.

```
SELECT student_id, student_name
FROM student
MINUS
SELECT student.student_id, student.student_name
FROM student
JOIN takes ON student.student_id = takes.student_id
WHERE takes.semester_year < 2010;
```

Results | Explain | Describe | Saved SQL | History

STUDENT_ID	STUDENT_NAME
19991	Brandt
23121	Chavez
55739	Sanchez
70557	Snow

4 rows returned in 0.01 seconds Download

6. Afficher tous les enseignants dont les noms commencent par E.

```
SELECT *
FROM teacher
WHERE teacher_name LIKE 'E%';
```

Results | Explain | Describe | Saved SQL | History

TEACHER_ID	TEACHER_NAME	DEPT_NAME	SALARY
22222	Einstein	Physics	95000
32343	El Said	History	60000

2 rows returned in 0.01 seconds Download

7. Afficher les salaires et les noms des enseignants qui perçoivent le quatrième salaire le plus élevé.

```
SELECT teacher_name, salary
FROM teacher T1
WHERE 3 = (
    SELECT COUNT(DISTINCT T2.salary)
    FROM teacher T2
    WHERE T2.salary > T1.salary
);
```

Results | Explain | Describe | Saved SQL | History

TEACHER_NAME	SALARY
Gold	87000

1 rows returned in 0.01 seconds Download

8. Afficher les noms et les salaires des trois enseignants qui perçoivent les salaires les moins élevés. Les afficher par ordre décroissant.

```sql
SELECT teacher_name, salary
FROM teacher T1
WHERE 2 >= (
    SELECT COUNT(DISTINCT T2.salary)
    FROM teacher T2
    WHERE T2.salary < T1.salary
)
ORDER BY salary DESC;
```

Results | Explain | Describe | Saved SQL | History

TEACHER_NAME	SALARY
Califieri	62000
El Said	60000
Mozart	40000

3 rows returned in 0.01 seconds Download

9. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la clause IN.

```sql
SELECT student_name
FROM student
WHERE student_id IN (
    SELECT student_id
    FROM takes
    WHERE semester = 'Fall' AND semester_year = 2009
);
```

Results | Explain | Describe | Saved SQL | History

STUDENT_NAME
Zhang
Shankar
Peltier
Levy
Williams
Brown
Bourikas

7 rows returned in 0.04 seconds Download

10. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la clause SOME.

```sql
SELECT student_name
FROM student
WHERE ('Fall', 2009) = SOME (
    SELECT semester, semester_year
    FROM takes
    WHERE student.student_id = takes.student_id
);
```

Results | Explain | Describe | Saved SQL | History

STUDENT_NAME
Zhang
Shankar
Peltier
Levy
Williams
Brown
Bourikas

7 rows returned in 0.03 seconds Download

11. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la jointure naturelle (NATURAL INNER JOIN).

```sql
SELECT DISTINCT student_name
FROM student
NATURAL INNER JOIN takes
WHERE semester = 'Fall' AND semester_year = 2009;
```

Results | Explain | Describe | Saved SQL | History

STUDENT_NAME
Brown
Zhang
Levy
Bourikas
Shankar
Peltier
Williams

7 rows returned in 0.03 seconds Download

12. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la clause EXISTS.

```sql
SELECT student_name
FROM student
WHERE EXISTS (
    SELECT 1
    FROM takes
    WHERE takes.student_id = student.student_id
      AND semester = 'Fall'
      AND semester_year = 2009
);
```

Results | Explain | Describe | Saved SQL | History

STUDENT_NAME
Zhang
Shankar
Peltier
Levy
Williams
Brown
Bourikas

7 rows returned in 0.03 seconds Download

13. Afficher toutes les paires des étudiants qui ont suivi au moins un cours ensemble.

```sql
SELECT s1.student_name, s2.student_name
FROM student s1
JOIN takes t1 ON s1.student_id = t1.student_id
JOIN student s2 ON 1=1
JOIN takes t2 ON s2.student_id = t2.student_id
WHERE t1.course_id = t2.course_id
  AND t1.sec_id = t2.sec_id
  AND t1.semester = t2.semester
  AND t1.semester_year = t2.semester_year
  AND s1.student_id < s2.student_id;
```

Results | Explain | Describe | Saved SQL | History

STUDENT_NAME	STUDENT_NAME
Zhang	Shankar
Zhang	Levy
Zhang	Williams
Zhang	Brown
Zhang	Bourikas
Zhang	Shankar
Shankar	Levy

14. Afficher pour chaque enseignant, qui a effectivement assuré un cours, le nombre total d'étudiants qui ont suivi ses cours. Si un étudiant a suivi deux cours différents avec le même enseignant, on le compte deux fois. Trier le résultat par ordre décroissant.

```
SELECT teacher.teacher_name, COUNT(*) AS total_etudiants
FROM takes
JOIN teaches ON takes.course_id = teaches.course_id
            AND takes.sec_id = teaches.sec_id
            AND takes.semester = teaches.semester
            AND takes.semester_year = teaches.semester_year
JOIN teacher ON teaches.teacher_id = teacher.teacher_id
GROUP BY teacher.teacher_name
ORDER BY total_etudiants DESC;
```

Results | Explain | Describe | Saved SQL | History

TEACHER_NAME	TOTAL_ETUDIANTS
Srinivasan	10
Brandt	3
Katz	2
Crick	2
Kim	1
Mozart	1
El Said	1
Einstein	1
Wu	1

9 rows returned in 0.07 seconds Download

15. Afficher pour chaque enseignant, même s'il n'a pas assuré de cours, le nombre total d'étudiants qui ont suivi ses cours. Si un étudiant a suivi deux fois un cours avec le même enseignant, on le compte deux fois. Trier le résultat par ordre décroissant.

```
SELECT teacher.teacher_name, COUNT(takes.course_id) AS total_etudiants
FROM teacher
LEFT JOIN teaches ON teacher.teacher_id = teaches.teacher_id
LEFT JOIN takes ON teaches.course_id = takes.course_id
               AND teaches.sec_id = takes.sec_id
               AND teaches.semester = takes.semester
               AND teaches.semester_year = takes.semester_year
GROUP BY teacher.teacher_name
ORDER BY total_etudiants DESC;
```

Results | Explain | Describe | Saved SQL | History

TEACHER_NAME	TOTAL_ETUDIANTS
Srinivasan	10
Brandt	3
Katz	2
Crick	2
El Said	1
Mozart	1
Wu	1
Kim	1
Einstein	1
Gold	0

16. Pour chaque enseignant, afficher le nombre total de grades A qu'il a attribué.

```
FROM teacher
LEFT JOIN teaches ON teacher.teacher_id = teaches.teacher_id
LEFT JOIN takes ON teaches.course_id = takes.course_id
               AND teaches.sec_id = takes.sec_id
               AND teaches.semester = takes.semester
               AND teaches.semester_year = takes.semester_year
WHERE takes.grade = 'A'
GROUP BY teacher.teacher_name
ORDER BY total_A DESC;
```

Results | Explain | Describe | Saved SQL | History

TEACHER_NAME	TOTAL_A
Srinivasan	4
Brandt	2
Crick	1

3 rows returned in 0.06 seconds Download

17. Afficher toutes les paires enseignant–élève où un élève a suivi le cours de l'enseignant, ainsi que le nombre de fois que cet élève a suivi un cours dispensé par cet enseignant.

```
SELECT teacher.teacher_name, student.student_name, COUNT(*) AS nb_cours
FROM teacher
JOIN teaches ON teacher.teacher_id = teaches.teacher_id
JOIN takes ON teaches.course_id = takes.course_id
          AND teaches.sec_id = takes.sec_id
          AND teaches.semester = takes.semester
          AND teaches.semester_year = takes.semester_year
JOIN student ON takes.student_id = student.student_id
GROUP BY teacher.teacher_name, student.student_name;
```

Results | Explain | Describe | Saved SQL | History

TEACHER_NAME	STUDENT_NAME	NB_COURS
Brandt	Shankar	1
Wu	Chavez	1
El Said	Brandt	1
Einstein	Peltier	1
Brandt	Brown	1
Srinivasan	Bourikas	2
Mozart	Sanchez	1
Kim	Aoi	1
Srinivasan	Shankar	3
Brandt	Williams	1

18. Afficher toutes les paires enseignant–élève où un élève a suivi au moins deux cours dispensés par l'enseignant en question.

```
SELECT teacher_student.teacher_name, teacher_student.student_name
FROM (
    SELECT teacher.teacher_name, student.student_name, COUNT(*) AS nb_cours
    FROM teacher
    JOIN teaches ON teacher.teacher_id = teaches.teacher_id
    JOIN takes ON teaches.course_id = takes.course_id
              AND teaches.sec_id = takes.sec_id
              AND teaches.semester = takes.semester
              AND teaches.semester_year = takes.semester_year
    JOIN student ON takes.student_id = student.student_id
    GROUP BY teacher.teacher_name, student.student_name
) teacher_student
WHERE teacher_student.nb_cours >= 2;
```

Results | Explain | Describe | Saved SQL | History

TEACHER_NAME	STUDENT_NAME
Srinivasan	Bourikas
Srinivasan	Shankar
Crick	Tanaka
Katz	Levy
Srinivasan	Zhang

5 rows returned in 0.04 seconds Download

Exercice 2:

Exercice n° 2

Donner la forme la plus avancée des schémas de relations suivants, munis de l'ensemble de dépendances fonctionnelles F. Si une relation n'est pas normalisée, la décomposer pour atteindre la forme normale la plus avancée.

1. $R(A, B, C)$ et $F = \{A \longrightarrow B; B \longrightarrow C\}$.
2. $R(A, B, C)$ et $F = \{A \longrightarrow C; A \longrightarrow B\}$.
3. $R(A, B, C)$ et $F = \{A, B \longrightarrow C; C \longrightarrow B\}$.

1. R(A,B,C) avec F={A→B,B→C}

- Fermeture de A :
  A+={A}→B→C→A+={A,B,C}
  A est clé candidate.
- Dépendance B→C viole BCNF (car B n'est pas une super-clé).

Décomposition en BCNF :

1. R1(B,C) avec B→C
2. R2(A,B) avec A→B

2. R(A,B,C) avec F={A→C,A→B}

- Fermeture de A :
  A+={A,B,C}
  A est clé candidate
- Toutes les dépendances ont une clé comme antécédent, donc pas de violation.

3. R(A,B,C) avec F={A,B→C;C→B}

- Fermeture de A,B :
  (A,B)+={A,B}→C→B
  → {A,B,C}, donc clé
- C→B viole BCNF car C n'est pas super-clé.

Décomposition en BCNF :

1. R1(C,B) avec C→B
2. R2(A,C)

Exercice 3:

1. Soit une relation $R(A, B, C, D, E)$ et un ensemble de dépendances fonctionnelles $F = \{A \longrightarrow B, C; C, D \longrightarrow E; B \longrightarrow D; E \longrightarrow A\}$. Déduire au moins 16 dépendances fonctionnelles, en utilisant le système d'Amstrong (règles de réflexivité, augmentation et transitivité).

Par réflexivité :

- A → A
- B → B
- C → C
- D → D
- E → E
- C,D → C
- C,D → D

Par augmentation :

- A → A,B (à partir de A → B)
- A → A,C (à partir de A → C)
- C,D → C,D,E (à partir de C,D → E)
- B → B,D (à partir de B → D)

➔ Par transitivité :

- A → D (car A → B et B → D)
- A → E (car A → C, A → D ⇒ C,D → E)
- E → B (car E → A et A → B)
- E → C (car E → A et A → C)
- E → D (car E → A → B → D)

2. Soit une relation $R(A, B, C, D, E, F)$ et un ensemble de dépendances fonctionnelles $F = \{A \longrightarrow B, C, D; B, C \longrightarrow D, E; B \longrightarrow D; D \longrightarrow A\}$.

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

   (b) Montrer que que $\{A, F\}$ est une super-clé.

   (c) Est-elle en BCNF (*Boyce–Codd Normal Form*) ? Si R n'est pas en BCNF, la décomposer pour atteindre cette forme normale.

2.a)
B+={A,B,C,D,E}

(A,B)+={A,B,C,D,E}

2.b)
(A,F)+={A,B,C,D,E,F}
Donc **{A,F} est bien une super-clé.**

2.c)
Dépendances dans F :

- A → B,C,D
- B,C → D,E
- B → D
- D → A

Pour la dépendance : **A → B,C,D**
Fermeture de A :

- A+={A,B,C,D}

Donc, **A n'est pas** une super-clé (car il manque E,F).

Pas en BCNF.

décomposition :

- Créer une relation R1(A,B,C,D) avec la dépendance A → B,C,D
- Créer une autre relation R2(A,E,F) avec les attributs restants

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

3. Soit une relation $R(A, B, C, D, E)$ que l'on décompose en deux relations

(a) $R_1(A, B, C)$ et $R_2(A, D, E)$. Montrer que cette décomposition est sans perte d'information.

(b) $R_1(A, B, C)$ $R_2(C, D, E)$. Montrer que cette décomposition est avec perte d'information.

3.a)
R1∩R2={A}
Et A→B,C,D donc A → A,B,C
Donc R1∩R2→R1
Donc la décomposition est **sans perte d'information**.

3.b)
R1∩R2={C}
C ne détermine pas R1 ni R2, donc la décomposition est **avec perte d'information**.

<u>Exercice 3:</u> (voir repository pour le code)

1. Écrire une procédure qui permet de prendre en paramètre une liste de dépendances fonctionnelles et les affiche.

```
if __name__ == "__main__":
    afficher_dependances(mydependencies)
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_
          {'A'} --> {'B'}
          {'A'} --> {'C'}
          {'G', 'C'} --> {'H'}
          {'G', 'C'} --> {'I'}
          {'B'} --> {'H'}
baba_sow@ing:~/Documents/BDA$
```

2. Écrire une procédure pour afficher un ensemble de relations.

```
if __name__ == "__main__":
    afficher_relations(myrelations)
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_
Voici les relations présentes :
        {'B', 'A', 'C', 'H', 'I', 'G'}
        {'X', 'Y'}
baba_sow@ing:~/Documents/BDA$
```

3. Écrire une fonction qui, prenant en paramètre un ensemble inputset, retourne tous ses sous-ensembles.

```
if __name__ == "__main__":
    print("Pour {'A', 'B', 'C'}")
    afficher_relations(ensemble_sous_ensembles({'A', 'B', 'C'}))
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_sow/Documents/BDA/TP_BDA_Baba_SOW/TP2/exo4.py
Pour {'A', 'B', 'C'}
Voici les relations présentes :
        {'C'}
        {'A'}
        {'B'}
        {'C', 'A'}
        {'C', 'B'}
        {'A', 'B'}
        {'C', 'A', 'B'}
baba_sow@ing:~/Documents/BDA$
```

4. Écrire une fonction qui, prenant en paramètre un ensemble de dépendances fonctionnelles F et un ensemble d'attributs K, retourne la fermeture (clôture) de K.

```
if __name__ == "__main__":
    fermeture_A = calculer_cloture_attributs(mydependencies, {'A'})
    print("La clôture de {A} dans mydependencies est :")
    print("\t", fermeture_A)
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_sow/Documents/BDA/TP_BDA_Baba_SOW/TP2/exo4.py
La clôture de {A} dans mydependencies est :
        {'B', 'C', 'A', 'H'}
baba_sow@ing:~/Documents/BDA$
```

5. Écrire une fonction qui, prenant en paramètre un ensemble de dépendances fonctionnelles F, retourne la clôture de F.

```python
if __name__ == "__main__":
    dependances = [
        [{'A'}, {'B'}],
        [{'B'}, {'C'}]
    ]

    cloture = generer_cloture_fonctionnelle(dependances)
    afficher_dependances(cloture)
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_sow/Documents/BDA/TP_BDA_Baba_SOW/TP2/exo4.py
        {'C'} --> {'C'}
        {'A'} --> {'C'}
        {'A'} --> {'A'}
        {'A'} --> {'B'}
        {'A'} --> {'A', 'C'}
        {'A'} --> {'C', 'B'}
        {'A'} --> {'A', 'B'}
        {'A'} --> {'A', 'C', 'B'}
        {'B'} --> {'C'}
        {'B'} --> {'B'}
        {'B'} --> {'C', 'B'}
        {'A', 'C'} --> {'A'}
        {'A', 'C'} --> {'C'}
        {'A', 'C'} --> {'B'}
        {'A', 'C'} --> {'A', 'C'}
        {'A', 'C'} --> {'A', 'B'}
        {'A', 'C'} --> {'C', 'B'}
        {'A', 'C'} --> {'A', 'C', 'B'}
        {'C', 'B'} --> {'C'}
        {'C', 'B'} --> {'B'}
        {'C', 'B'} --> {'C', 'B'}
        {'A', 'B'} --> {'C'}
        {'A', 'B'} --> {'A'}
        {'A', 'B'} --> {'B'}
        {'A', 'B'} --> {'A', 'C'}
        {'A', 'B'} --> {'C', 'B'}
        {'A', 'B'} --> {'A', 'B'}
        {'A', 'B'} --> {'A', 'C', 'B'}
        {'A', 'C', 'B'} --> {'A'}
        {'A', 'C', 'B'} --> {'C'}
        {'A', 'C', 'B'} --> {'B'}
        {'A', 'C', 'B'} --> {'A', 'C'}
        {'A', 'C', 'B'} --> {'A', 'B'}
        {'A', 'C', 'B'} --> {'C', 'B'}
        {'A', 'C', 'B'} --> {'A', 'C', 'B'}
baba_sow@ing:~/Documents/BDA$
```

6. Écrire une fonction qui, prenant en paramètre un ensemble de dépendances fonctionnelles F et deux ensembles d'attributs α et β, retourne vrai si α détermine fonctionnellement β.

```
if __name__ == "__main__":
    dependances = [
        [{'A'}, {'B'}],
        [{'B'}, {'C'}]
    ]

    resultat = verifier_dependance(dependances, {'A'}, {'C'})
    if resultat:
        print("La dépendance A → C est valide.")
    else:
        print("La dépendance A → C n'est pas valide.")
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_
La dépendance A → C est valide.
baba_sow@ing:~/Documents/BDA$
```

7. Écrire une fonction qui, prenant en paramètre un ensemble de dépendances fonctionnelles F, une relation R et un ensemble d'attributs K, retourne vrai si K est une super-clé.

```
if __name__ == "__main__":
    dependance = [
        [{'A'}, {'B'}],
        [{'B'}, {'C'}]
    ]
    relation = {'A', 'B', 'C'}

    resultat = est_super_cle(dependance, relation, {'A'})
    if resultat:
        print("Le groupe {A} est une super-clé de la relation.")
    else:
        print("Le groupe {A} n'est pas une super-clé de la relation.")
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_sow/Documents/BD
Le groupe {A} est une super-clé de la relation.
baba_sow@ing:~/Documents/BDA$
```

8. Écrire une fonction qui, prenant en paramètre un ensemble de dépendances fonctionnelles F, une relation R et un ensemble d'attributs K, retourne vrai si K est une clé candidate.

```python
if __name__ == "__main__":
    dependances = [
        [{'A'}, {'B'}],
        [{'B'}, {'C'}]
    ]
    relation = {'A', 'B', 'C'}

    resultat = est_cle_candidate(dependances, relation, {'A'})
    if resultat:
        print("Le groupe {A} est une clé candidate de la relation.")
    else:
        print("Le groupe {A} n'est pas une clé candidate de la relation.")
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba
Le groupe {A} est une clé candidate de la relation.
baba_sow@ing:~/Documents/BDA$
```

9. Écrire une fonction qui, prenant en paramètre une relation R et un ensemble de dépendances fonctionnelles F, retourne la liste de toutes les clés candidates.

```python
if __name__ == "__main__":
    cles = trouver_cles_candidates(mydependencies, myrelations[0])
    print("Clés candidates trouvées pour la relation", myrelations[0], ":")
    afficher_relations(cles)
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_sow/Documents/BDA/TP_BD
Clés candidates trouvées pour la relation {'B', 'I', 'G', 'C', 'A', 'H'} :
Voici les relations présentes :
        {'A', 'G'}
baba_sow@ing:~/Documents/BDA$
```

10. Écrire une fonction qui, prenant en paramètre une relation R et un ensemble de dépendances fonctionnelles F, retourne la liste de toutes les super-clés.

```python
if __name__ == "__main__":
    for relation in myrelations:
        print("Pour la relation :", relation)
        super_cles = trouver_super_cles(mydependencies, relation)
        print("Super-clés trouvées :")
        afficher_relations(super_cles)
        print("-" * 40)
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_
Pour la relation : {'G', 'A', 'C', 'H', 'B', 'I'}
Super-clés trouvées :
Voici les relations présentes :
        {'G', 'A'}
        {'G', 'A', 'C'}
        {'G', 'A', 'H'}
        {'G', 'A', 'B'}
        {'G', 'A', 'I'}
        {'G', 'A', 'C', 'H'}
        {'G', 'A', 'B', 'C'}
        {'G', 'I', 'A', 'C'}
        {'G', 'A', 'B', 'H'}
        {'G', 'A', 'I', 'H'}
        {'G', 'I', 'A', 'B'}
        {'C', 'B', 'G', 'H', 'A'}
        {'C', 'I', 'G', 'H', 'A'}
        {'C', 'B', 'I', 'G', 'A'}
        {'B', 'I', 'G', 'H', 'A'}
        {'C', 'B', 'I', 'G', 'H', 'A'}
----------------------------------------
Pour la relation : {'X', 'Y'}
Super-clés trouvées :
Voici les relations présentes :
        {'X', 'Y'}
----------------------------------------
baba_sow@ing:~/Documents/BDA$
```

11. Écrire une fonction qui, prenant en paramètre un ensemble de dépendances fonctionnelles F et une relation R, retourne une clé candidate.

```
if __name__ == "__main__":
    for relation in myrelations:
        print("Pour la relation :", relation)
        une_cle = trouver_une_cle_candidate(mydependencies, relation)
        print("Une clé candidate trouvée :", une_cle)
        print("-" * 40)
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_sow
Pour la relation : {'B', 'I', 'H', 'C', 'G', 'A'}
Une clé candidate trouvée : {'G', 'A'}
----------------------------------------
Pour la relation : {'X', 'Y'}
Une clé candidate trouvée : {'X', 'Y'}
----------------------------------------
baba_sow@ing:~/Documents/BDA$
```

12. Écrire une fonction qui, prenant en paramètre un ensemble de dépendances fonctionnelles F et une relation R, retourne vrai si cette relation est en BCNF.

```
if __name__ == "__main__":
    for relation in myrelations:
        print("Pour la relation :", relation)
        if est_relation_bcnf(mydependencies, relation):
            print("Cette relation est en BCNF.")
        else:
            print("Cette relation n'est pas en BCNF.")
        print("-" * 40)
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba_
Pour la relation : {'G', 'I', 'A', 'C', 'B', 'H'}
Cette relation n'est pas en BCNF.
----------------------------------------
Pour la relation : {'X', 'Y'}
Cette relation est en BCNF.
----------------------------------------
baba_sow@ing:~/Documents/BDA$
```

13. Écrire une fonction qui, prenant en paramètre un ensemble de dépendances fonctionnelles F et un ensemble de relations T, retourne vrai si le schéma défini par ces relations est en BCNF.

```
if __name__ == "__main__":
    if est_schema_bcnf(mydependencies, myrelations):
        print("Le schéma est en BCNF.")
    else:
        print("Le schéma n'est pas en BCNF.")
```

```
baba_sow@ing:~/Documents/BDA$ /bin/pyth
Le schéma n'est pas en BCNF.
baba_sow@ing:~/Documents/BDA$
```

14. Écrire une fonction qui, prenant en paramètre un ensemble de dépendances fonctionnelles F et un ensemble de relations T, implémente l'algorithme de décomposition en BCNF.

```
if __name__ == "__main__":
    resultat_bcnf = decomposer_schema_bcnf(mydependencies, myrelations)
    print("Relations obtenues après décomposition BCNF :")
    afficher_relations(resultat_bcnf)
```

```
baba_sow@ing:~/Documents/BDA$ /bin/python3 /home/baba
Relations obtenues après décomposition BCNF :
Voici les relations présentes :
        {'Y', 'X'}
        {'A', 'I', 'G'}
        {'H', 'B'}
        {'C', 'A', 'B'}
baba_sow@ing:~/Documents/BDA$
```