

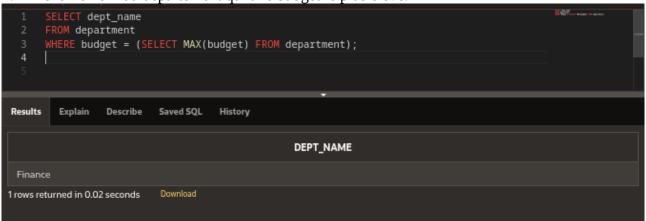


# TP2 - BDA

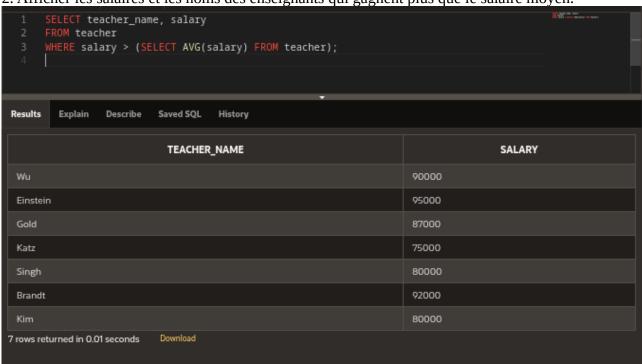
Baba SOW ING – INFO2

#### Exercic1:

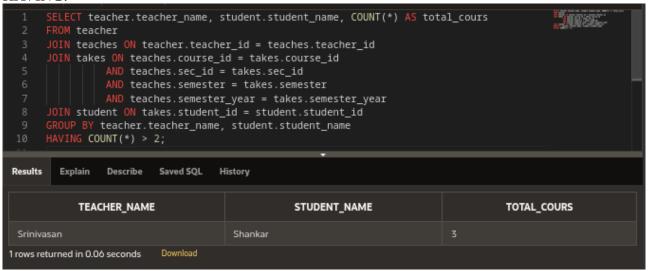
1. Afficher le nom du département qui a le budget le plus élevé.



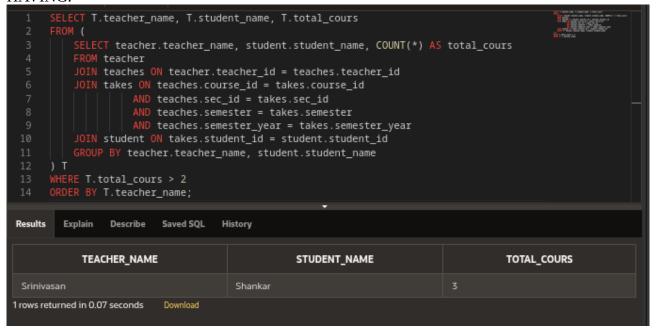
2. Afficher les salaires et les noms des enseignants qui gagnent plus que le salaire moyen.



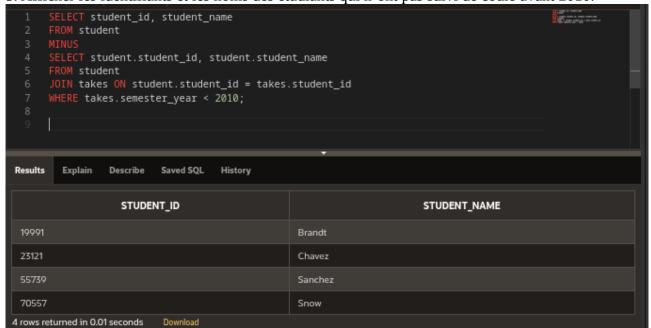
3. Pour chaque enseignant, afficher tous les étudiants qui ont suivi plus de deux cours dispensés par cet enseignant ainsi que le nombre total de cours suivis par chaque étudiant, en utilisant la clause HAVING.



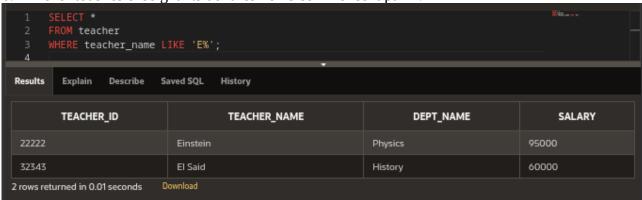
4. Pour chaque enseignant, afficher tous les étudiants qui ont suivi plus de deux cours dispensés par cet enseignant ainsi que le nombre total de cours suivis par chaque étudiant, sans utiliser la clause HAVING.



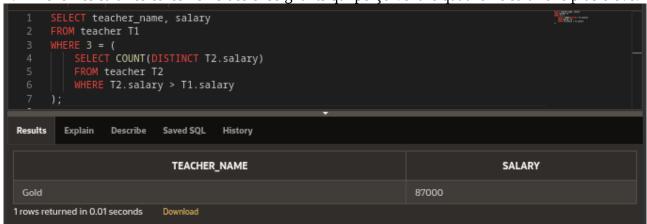
5. Afficher les identifiants et les noms des étudiants qui n'ont pas suivi de cours avant 2010.



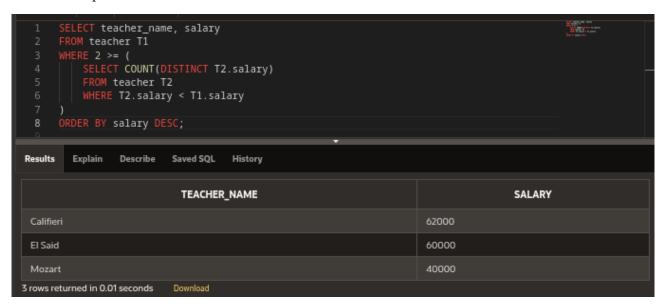
6. Afficher tous les enseignants dont les noms commencent par E.



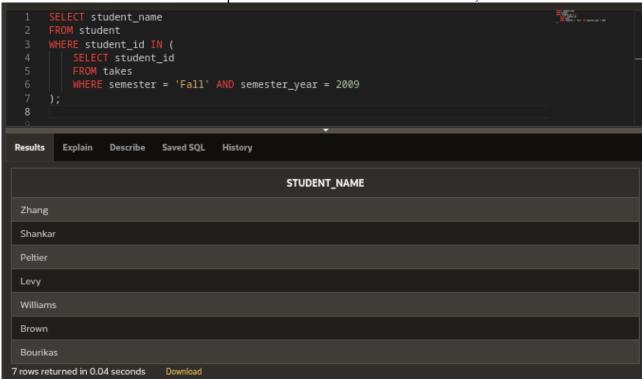
7. Afficher les salaires et les noms des enseignants qui perçoivent le quatrième salaire le plus élevé.



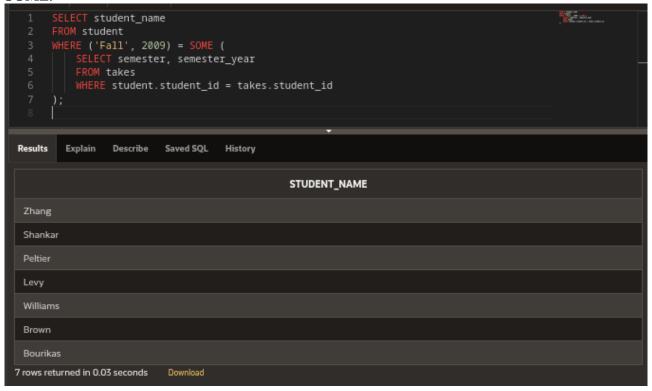
8. Afficher les noms et les salaires des trois enseignants qui perçoivent les salaires les moins élevés. Les afficher par ordre décroissant.



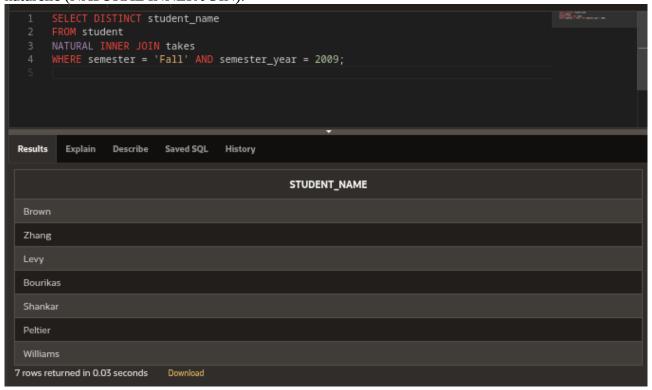
9. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la clause IN.



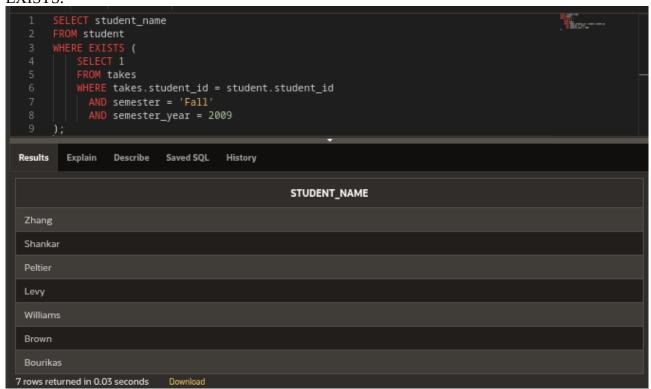
10. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la clause SOME.



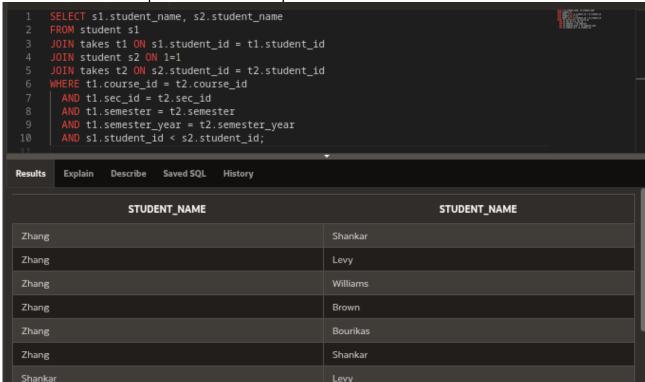
11. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la jointure naturelle (NATURAL INNER JOIN).



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



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



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

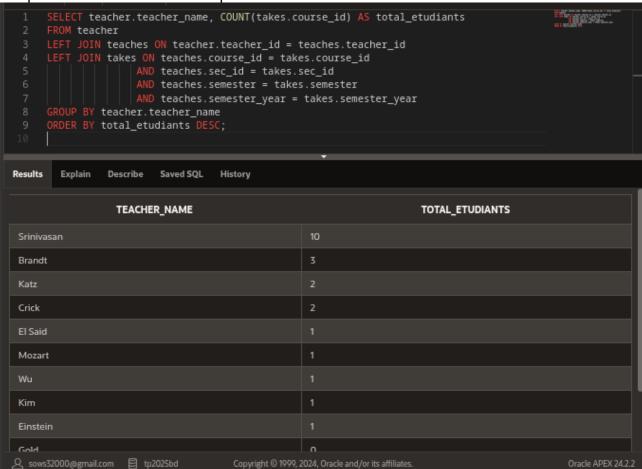
AND takes.sec\_id = teaches.sec\_id

AND takes.semester = teaches.semester

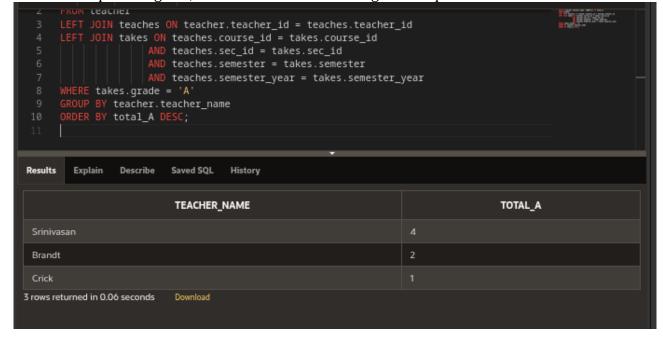
AND takes.semester\_year = teaches.semester\_year JOIN teacher ON teaches.teacher\_id = teacher.teacher\_id GROUP BY teacher.teacher\_name ORDER BY total\_etudiants DESC; Explain Describe Saved SQL History Results TEACHER\_NAME TOTAL\_ETUDIANTS Srinivasan Brandt Crick Mozart El Said Einstein 9 rows returned in 0.07 seconds Download Sows32000@gmail.com tp2025bd Copyright © 1999, 2024, Oracle and/or its affiliate

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.



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

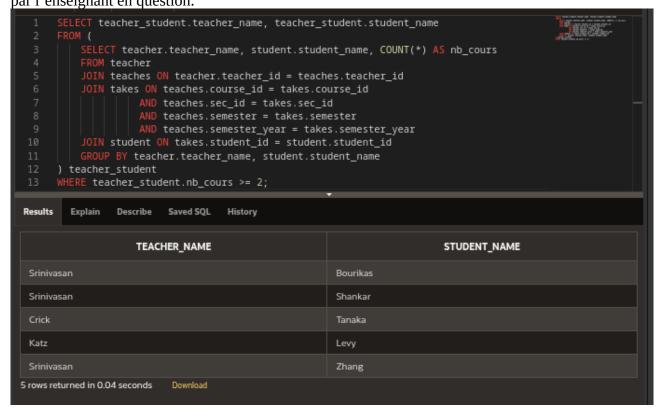


17. Afficher toutes les paires enseignant—élève où un élève a suivi le cours de l'enseignant, ainsi que

le nombre de fois que cet élève a suivi un cours dispensé par cet enseignant.

1 SELECT teacher.teacher_name, student.student_name, COUNT(*) AS nb_cours 2 FROM teacher 3 JOIN teaches ON teacher.teacher_id = teaches.teacher_id 4 JOIN takes ON teaches.course_id = takes.course_id 5 AND teaches.sec_id = takes.sec_id 6 AND teaches.semester = takes.semester 7 AND teaches.semester_year = takes.semester_year 8 JOIN student ON takes.student_id = student.student_id 9 GROUP BY teacher.teacher_name, student.student_name; 10  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	
Kim	Aoi	
Srinivasan	Shankar	
Brandt	Williams	1
🙎 sows32000@gmail.com 🛢 tp2025bd Copyright © 1999, 2024, Oracle and/or its affiliates. Oracle APEX 24.2.2		

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



#### **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 $F=\{A \rightarrow B,B \rightarrow C\}$

• Fermeture de A:

$$A += \{A\} \rightarrow B \rightarrow C \rightarrow A += \{A,B,C\}$$

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  $B \rightarrow C$
- 2. R2(A,B) avec  $A \rightarrow B$

## 2. R(A,B,C) avec $F=\{A \rightarrow C,A \rightarrow B\}$

• 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 $F=\{A,B \rightarrow C;C \rightarrow B\}$

• Fermeture de A,B:

$$(A,B)+=\{A,B\} \rightarrow C \rightarrow B$$

- → {A,B,C}, donc clé
- C → B viole BCNF car C n'est pas super-clé.

#### Décomposition en BCNF :

- 1. R1(C,B) avec  $C \rightarrow B$
- 2. R2(A,C)

## **Exercice 3:**

Soit une relation R(A, B, C, D, E) et un ensemble de dépendances fonctionnelles
 F = {A → B, C; C, D → E; B → D; E → 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
- $D \rightarrow D$
- E → E
- C,D → C
- $C,D \rightarrow D$

### Par augmentation:

- $A \rightarrow A,B$  (à partir de  $A \rightarrow B$ )
- $A \rightarrow A,C$  (à partir de  $A \rightarrow C$ )
- $C,D \rightarrow C,D,E$  (à partir de  $C,D \rightarrow E$ )
- B  $\rightarrow$  B,D (à partir de B  $\rightarrow$  D)

#### → Par transitivité :

- $A \rightarrow D (car A \rightarrow B et B \rightarrow D)$
- $A \rightarrow E (car A \rightarrow C, A \rightarrow D \Rightarrow C, D \rightarrow E)$
- $E \rightarrow B (car E \rightarrow A et A \rightarrow B)$
- $E \rightarrow C (car E \rightarrow A et A \rightarrow C)$
- $E \rightarrow D (car E \rightarrow A \rightarrow B \rightarrow D)$

- 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.c)

Dépendances dans F:

- $A \rightarrow B,C,D$
- $B,C \rightarrow D,E$
- $B \rightarrow D$
- $D \rightarrow A$

Pour la dépendance :  $A \rightarrow B,C,D$ 

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)

- Soit une relation R(A, B, C, D, E) que l'on décompose en deux relations
  - (a) R<sub>1</sub>(A, B, C) et R<sub>2</sub>(A, D, E). Montrer que cette décomposition est sans perte d'information.
  - (b) R<sub>1</sub>(A, B, C) R<sub>2</sub>(C, D, E). Montrer que cette décomposition est avec perte d'information.

3.a)  $R1 \cap R2 = \{A\}$ Et  $A \rightarrow B,C,D$  donc  $A \rightarrow A,B,C$ Donc  $R1 \cap R2 \rightarrow R1$ 

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**.

## **Exercice 3:** (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)
```

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

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

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

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

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)
```

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

```
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'\} \longrightarrow \{'A'\}
      \{'A'\} \longrightarrow \{'B'\}
      {'A'} --> {'A', 'C'}
      {'A'} --> {'C', 'B'}
      {'A'} --> {'A', 'B'}
      {'A'} --> {'A', 'C', 'B'}
      {'B'} --> {'C'}
      \{'B'\} \longrightarrow \{'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'\} \longrightarrow \{'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  $\alpha$  et  $\beta$ , retourne vrai si  $\alpha$  détermine fonctionnellement  $\beta$ .

```
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.

$\footnote{\text{baba_sow@ing:~/Documents/BDA$}}$
$\footnote{\text{CodeSize}}$
$\footnote{\
```

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/BDA$
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.

```
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.

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

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.

```
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'}

$\text{baba_sow@ing:~/Documents/RDA$}
```

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/pytho
        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)
```