

TP 2 BDA

Exercice 1 :

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

```
1  SELECT dept_name
2  FROM department
3  WHERE budget in ( SELECT max ( budget ) FROM department ) ;
```

Results Explain Describe Saved SQL History

DEPT_NAME
Finance

1 rows returned in 0.01 seconds [Download](#)

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

```
1  SELECT A.name , A.salary FROM teacher A
2  WHERE A.salary > (SELECT avg(B.salary ) FROM teacher B );
```

Results Explain Describe Saved SQL History

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

7 rows returned in 0.02 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.

```

1 SELECT teacher . name , student . name , count (*)
2 FROM teacher , student , takes , teaches
3 WHERE teacher.id = teaches.id and student.id = takes.id
4 and takes.course_id=teaches.course_id
5 and takes.sec_id =teaches.sec_id
6 and takes.semester =teaches.semester
7 and takes.year = teaches.year
8 GROUP BY teacher . name , student . name HAVING count(*) >= 2;

```

NAME	NAME	COUNT(*)
Srinivasan	Bourikas	2
Srinivasan	Shankar	3
Crick	Tanaka	2
Katz	Levy	2
Srinivasan	Zhang	2

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.

```

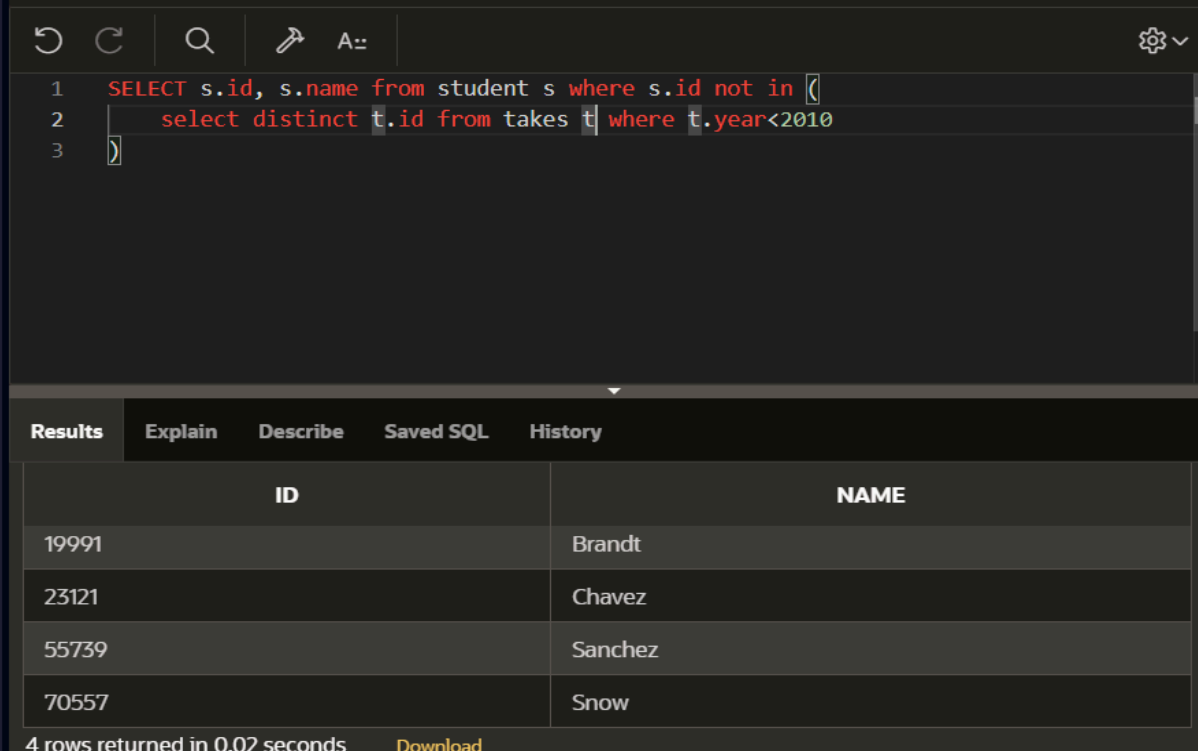
1 SELECT T.teachername , T.studentname , T.totalcount
2 FROM ( SELECT teacher . name as teachername , student . name as
3 studentname , count (*) as totalcount
4 FROM teacher , student , takes , teaches
5 WHERE teacher.id=teaches.id
6 and student.id = takes.id
7 and takes.course_id = teaches.course_id
8 and takes.sec_id = teaches . sec_id
9 and takes.semester= teaches . semester
10 and takes . year=teaches . year
11 GROUP BY teacher.name , student.name ) T
12 WHERE T.totalcount >= 2

```

TEACHERNAME	STUDENTNAME	TOTALCOUNT
Katz	Levy	2
Srinivasan	Bourikas	2
Srinivasan	Shankar	3
Srinivasan	Zhang	2

5 rows returned in 0.05 seconds [Download](#)

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

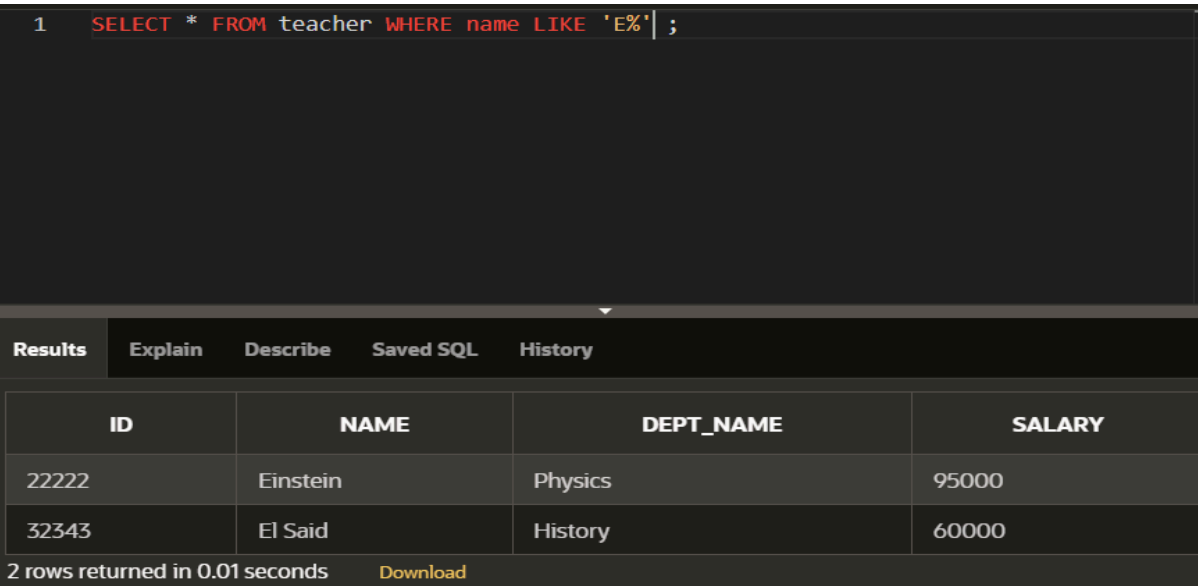


```
1 SELECT s.id, s.name from student s where s.id not in (
2     select distinct t.id from takes t where t.year<2010
3 )
```

ID	NAME
19991	Brandt
23121	Chavez
55739	Sanchez
70557	Snow

4 rows returned in 0.02 seconds [Download](#)

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



```
1 SELECT * FROM teacher WHERE name LIKE 'E%';
```

ID	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.

```
1  SELECT name, salary
2  FROM teacher T1
3  WHERE 3 = (
4      SELECT COUNT(DISTINCT T2.salary)
5      FROM teacher T2
6      WHERE T2.salary > T1.salary
7  );
8
```

Results Explain Describe Saved SQL History

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.

```
1  SELECT T1 . name , T1 . salary FROM teacher T1
2  WHERE 2 >= (
3      SELECT COUNT ( DISTINCT T2 . salary ) FROM teacher T2
4      WHERE T2 . salary < T1 . salary )
5  ORDER BY T1 . salary DESC ;|
```

Results Explain Describe Saved SQL History

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.

```
1  SELECT S. name
2  FROM student S
3  WHERE ( 'Fall' , 2009) IN (SELECT semester , year
4  FROM takes WHERE takes . id = S. id );
```

Results	Explain	Describe	Saved SQL	History
NAME				
Zhang				
Shankar				
Peltier				
Levy				
Williams				
Brown				
Bourikas				
7 rows returned in 0.03 seconds Download				

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

```
1  SELECT S. name
2  FROM student S
3  WHERE ( 'Fall' , 2009) = SOME
4  ( SELECT semester , year FROM takes WHERE takes . id = S. id );
```

Results	Explain	Describe	Saved SQL	History
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).

```
1 SELECT DISTINCT student . name
2 FROM takes NATURAL INNER JOIN student
3 WHERE takes . semester = 'Fall' AND takes . year = 2009;
```

Results Explain Describe Saved SQL History

NAME
Brown
Zhang
Levy
Bourikas
Shankar
Peltier
Williams

7 rows returned in 0.01 seconds [Download](#)

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

```
1 SELECT name FROM student
2 WHERE EXISTS ( SELECT * FROM takes
3 WHERE takes . id = student . id AND semester = 'Fall' AND year = 2009 ) ;
```

Results Explain Describe Saved SQL History

NAME
Zhang
Shankar
Peltier
Levy
Williams
Brown
Bourikas

7 rows returned in 0.02 seconds [Download](#)

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

```

1  SELECT DISTINCT S1.name , S2.name
2  FROM student s1
3  JOIN takes t1 on s1.ID = t1.ID
4  JOIN section sec1 on t1.course_id = sec1.course_id and t1.sec_id=sec1.sec_id
5  JOIN takes t2 on sec1.course_id = t2.course_id and sec1.sec_id = t2.sec_id
6  JOIN student s2 on t2.ID=s2.ID
7  WHERE s1.ID < s2.ID;

```

Results	Explain	Describe	Saved SQL	History
NAME		NAME		
Shankar		Levy		
Shankar		Williams		
Levy		Brown		
Levy		Bourikas		
More than 10 rows available. Increase rows selector to view more rows.				
10 rows returned in 0.08 seconds Download				

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.

```

1  SELECT teacher . name , count (*)
2  FROM ( takes INNER JOIN teaches USING (course_id,sec_id,semester,year))
3  INNER JOIN teacher ON teaches . id = teacher . id
4  GROUP BY teacher . name , teacher . id
5  ORDER BY count (*) DESC ;

```

Results	Explain	Describe	Saved SQL	History
NAME			COUNT(*)	
Katz			2	
Einstein			1	
El Said			1	
Mozart			1	
Wu			1	
Kim			1	
9 rows returned in 0.04 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.

```
1 SELECT teacher . name , count (course_id)
2 FROM ( takes INNER JOIN teaches USING (course_id,sec_id,semester,year) )
3 RIGHT OUTER JOIN teacher ON teaches . id = teacher . id
4 GROUP BY teacher . name , teacher . id
5 ORDER BY count ( course_id ) DESC ;
```

Results	Explain	Describe	Saved SQL	History
NAME		COUNT(COURSE_ID)		
Srinivasan		10		
Brandt		3		
Katz		2		
Crick		2		
El Said		1		
Einstein		1		
Kim		1		

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

```
1 WITH mytakes ( id , course_id , sec_id , semester , year , grade )
2 AS ( SELECT id ,course_id , sec_id , semester , year , grade
3 FROM takes
4 WHERE grade = 'A' )
5 SELECT teacher . name , count ( course_id )
6 FROM ( mytakes INNER JOIN teaches USING (course_id,sec_id,semester,year))
7 RIGHT OUTER JOIN teacher ON teaches . id = teacher . id
8 GROUP BY teacher . name , teacher . id
9 ORDER BY count (course_id) DESC ;
```

Results	Explain	Describe	Saved SQL	History
NAME		COUNT(COURSE_ID)		
Srinivasan		4		
Brandt		2		
Crick		1		
Califieri		0		

17. Afficher toutes les paires enseignants-élèves 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 . name , student . name , count (*)
2 FROM ( teacher NATURAL JOIN teaches )
3 INNER JOIN
4 ( takes NATURAL JOIN student )
5 USING( course_id , sec_id , semester , year )
6 GROUP BY teacher . name , student . name ;
```

Results	Explain	Describe	Saved SQL	History
NAME		NAME	COUNT(*)	
Brandt		Shankar	1	
Wu		Chavez	1	
El Said		Brandt	1	
Einstein		Peltier	1	
Brandt		Brown	1	

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

Exercice 2 :**1. $R(A, B, C)$ et $F = \{A \rightarrow B; B \rightarrow C\}$.**

- **Clé candidate** : A est une clé candidate ($A \rightarrow B$ et $B \rightarrow C$ donc $A \rightarrow C$ aussi).
- **Forme Normale Actuelle** :
 - **1NF** (on suppose que les attributs contiennent des valeurs atomiques).
 - **2NF** (pas de dépendance partielle car A est la seule clé).
 - **pas en 3NF** ($B \rightarrow C$ est une dépendance transitive, C dépend d'un attribut non clé B).
- **Décomposition en 3NF** :
 - $R_1(A, B)$ avec $A \rightarrow B$
 - $R_2(B, C)$ avec $B \rightarrow C$

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

- **Clé candidate** : A est une clé candidate (A détermine B et C).
- **Forme Normale Actuelle** :
 - **1NF**
 - **2NF** (pas de dépendance partielle car A est la seule clé).
 - **3NF** (toutes les dépendances fonctionnelles sont respectées car C et B dépendent directement de la clé).
 - **BCNF** (chaque dépendance est de la forme clé \rightarrow attribut).

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

- **Clé candidate** : A, B est une super-clé, mais $C \rightarrow B$ implique que $\{A, C\}$ est aussi une clé candidate.
- **Forme Normale Actuelle** :
 - **1NF**
 - **2NF** (pas de dépendance partielle).
 - **pas en 3NF** ($C \rightarrow B$ est une dépendance transitive).
- **Décomposition en 3NF** :
 - $R_1(A, C)$ avec $A \rightarrow C$
 - $R_2(C, B)$ avec $C \rightarrow B$

Exercice 3 :

1. $R(A, B, C, D, E)$ et $F = \{A \rightarrow B, C ; C, D \rightarrow E ; B \rightarrow D ; E \rightarrow A\}$.

- **Réflexivité (si $Y \subseteq X$ alors $X \rightarrow Y$)**
 - $A \rightarrow A$
 - $B \rightarrow B$
 - $C \rightarrow C$
 - $D \rightarrow D$
 - $E \rightarrow E$
- **Augmentation (si $X \rightarrow Y$ alors $XZ \rightarrow YZ$)**
 - $A \rightarrow B, C$ alors $A, D \rightarrow B, C, D$
 - $B \rightarrow D$ alors $B, C \rightarrow D, C$
 - $C, D \rightarrow E$ alors $C, D, A \rightarrow E, A$
 - ...
- **Transitivité (si $X \rightarrow Y$ et $Y \rightarrow Z$ alors $X \rightarrow Z$)**
 - $A \rightarrow B, C$ et $B, C \rightarrow D, C$ alors $A \rightarrow D, C$
 - $C, D \rightarrow E$ et $E \rightarrow A$ alors $C, D \rightarrow A$
 - ...

2.

- **Fermeture de B et $\{A, B\}$**
 - $B \rightarrow D$, donc $B^+ = \{B, D\}$
 - $A \rightarrow B, C, D$ donc $\{A, B\}^+ = \{A, B, C, D\}$
 - $A, B, C, D \rightarrow E$ et $E \rightarrow A$ donc $A, B, C, D, E \rightarrow A, B, C, D, E$
 - donc $\{A, B\}$ est une super-clé

3.

- **$R_1(A, B, C)$ et $R_2(A, D, E)$**
 - A est une clé de l'un des deux schémas (R_1) \Rightarrow Décomposition sans perte
- **$R_1(A, B, C)$ et $R_2(C, D, E)$**
 - mais C seul n'est pas une clé. \Rightarrow Décomposition avec perte d'information

Exercice 4 :

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

```

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

[3] mydependencies = [
    [ {'A'}, {'B'} ], #A->B
    [ {'A'}, {'C'} ], #A->C
    [ {'C', 'G'}, {'H'} ], #CG ->H
    [ {'C', 'G'}, {'I'} ], #CG ->I
    [ {'B'}, {'H'} ] #B->H
]

[5] def printDependencies ( F: "list of dependencies"):
    for alpha , beta in F:
        print ("\t", alpha , " --> ", beta )

```

2. Écrire une procédure qui permet de prendre en paramètre un ensemble de relations T et les affiche.

```

+ Code + Texte
0 s [6] def printRelations (T: "list of relations") :
    for R in T:
        print ("\t", R )

```

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

```

0 s [7] import itertools

0 s def powerSet ( inputset : " set "):
    _result = []
    for r in range (1 , len ( inputset ) +1) :
        _result += map (set , itertools . combinations ( inputset , r))
    return _result

```

4. Écrire une fonction qui permet, étant donné un ensemble de dépendances fonctionnelles F et un ensemble d'attributs K , de retourner la fermeture (clôture) de K .

```
[9] def computeAttributeClosure (F: " list of dependencies ", K: " set of attributes "):
    K_plus , size = set (K) , 0
    while size != len ( K_plus ):
        size = len ( K_plus )
        for alpha , beta in F:
            if alpha . issubset ( K_plus ):
                K_plus . update ( beta )
    return K_plus
```

5. Écrire une fonction qui permet, étant donné un ensemble de dépendances fonctionnelles F , de retourner la clôture de F .

```
[12] def computeDependenciesClosure (F : " list of dependencies "):
    R = set ()
    for alpha , beta in F:
        R. update ( alpha | beta )
    F_plus = []
    for K in powerSet (R) :
        for beta in powerSet ( computeAttributeClosure (F , K) ) :
            F_plus . append ( [K , beta ] )
    return F_plus
```

6. Écrire une fonction qui permet, étant donné un ensemble de dépendances fonctionnelles F et deux ensembles d'attributs α et β , de retourner vrai si α détermine fonctionnellement β .

```
[13] def isDependency (F : " list of dependencies ", alpha : " set of attributes ", beta : "set of attributes "):
    return beta . issubset ( computeAttributeClosure (F , alpha ) )
```

7. Écrire une fonction qui permet, étant donné un ensemble de dépendances fonctionnelles F , une relation R et un ensemble d'attributs K , de retourner vrai si K est une super-clé.

```
def isSuperKey (F: " list of dependencies ", R: " set defining a relation ", K: " set of attributes "):
    return R. issubset ( computeAttributeClosure (F , K) )
```

8. Écrire une fonction qui permet, étant donnée un ensemble de dépendances fonctionnelles F , une relation R et un ensemble d'attributs K , de retourner vrai si K est une clé candidate

```
def isCandidateKey (F: " list of dependencies ", R: " set defining a relation ", K: "set of attributes "):
    if not isSuperKey (F , R , K): return False
    for A in K:
        _K1 = set (K)
        _K1 . discard (A)
        if isSuperKey (F , R , _K1 ): return False
    return True
```

9. Ecrire une fonction qui, étant donné une relation R et un ensemble de dépendances fonctionnelles F, de retourner la liste de toutes les clés candidates

```
0 s [16] def computeAllCandidateKeys (F: " list of dependencies ", R: " set defining a relation "):
    result = []
    for K in powerSet (R) :
        if isCandidateKey (F , R , K ):
            result.append (K)
    return result
```

10. Écrire une fonction qui, étant donnée une relation R et un ensemble de dépendances fonctionnelles F, de retourner la liste de toutes les super-clés.

```
0 s [17] def computeAllSuperKeys (F: " list of dependencies ", R: " set defining a relation "):
    result = []
    for K in powerSet (R) :
        if isSuperKey (F , R , K):
            result . append (K)
    return result
```

11. Écrire une fonction qui permet, étant donné un ensemble de dépendances fonctionnelles F et une relation R, de retourner une clé candidate.

```
0 s [18] def computeOneCandidateKey ( F: " list of dependencies ", R: " set defining a relation "):
    K = set (R )
    while not isCandidateKey (F , R , K ):
        for A in K:
            if isSuperKey ( F , R , K. difference ({ A } ) ):
                K. remove (A)
                break
    return K
```

12. Écrire une fonction qui permet, étant donnée une relation R et un ensemble de dépendances fonctionnelles F, de retourner vrai si cette relation est en BCNF.

```
0 s [19] def isBCNFRelation (F: " list of dependencies ", R: " set defining a relation "):
    for K in powerSet (R) :
        K_plus = computeAttributeClosure (F , K)
        Y = K_plus . difference (K )
        if not R . issubset ( K_plus ) and not Y. isdisjoint (R):
            return False , [ K , Y & R ]

    return True , [ {} , {} ]
```

13. Écrire une fonction qui permet, étant donné un ensemble de relations T et une liste de dépendances fonctionnelles F, de retourner vrai si le schéma défini par ces relations est en BCNF.

```
0 s [20] def isBCNFRelations (F: " list of dependencies ", T: " list of relations "):
    for R in T:
        if isBCNFRelation (F , R ) == False :
            return False , R
    return True , {}
```

14. Écrire une fonction qui permet, étant donné un ensemble de dépendances fonctionnelles F et un ensemble de relations T, d'implémenter l'algorithme de décomposition en BCNF, vu en cours.

```
✓ [21] def computeBCNFDecomposition (F: " list of dependencies ", T: " list of relations "):  
0 s OUT , size = list (T) , 0  
while size != len ( OUT ):  
    size = len ( OUT )  
    for R in OUT :  
        _isR_BCNF , [ alpha , beta ] = isBCNFRelation (F , R )  
        if _isR_BCNF == False :  
            if alpha | beta not in OUT :  
                OUT . append ( alpha | beta )  
            if R. difference ( beta ) not in OUT :  
                OUT . append ( R. difference ( beta ) )  
            OUT . remove (R)  
            break  
    return OUT
```