Rapport TP 2 – Base de données avancées

Nom : Lyes Ahfir
Date : 30 avril 2025

Exercice 1 : Suite du TP1

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

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

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

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

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.

```
SELECT teacher.name, student.name, count(*)
FROM teacher, student, takes, teaches
WHERE teacher.id = teaches.id AND student.id = takes.id AND
      takes.course_id = teaches.course_id AND
      takes.sec_id = teaches.sec_id AND
      takes.semester = teaches.semester AND
      takes.year = teaches.year
GROUP BY teacher.name, student.name
HAVING count(*) >= 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.

```
SELECT T.teachername, T.studentname, T.totalcount
FROM (SELECT teacher.name AS teachername, student.name AS studentname,
    count(*) as totalcount
       FROM teacher, student, takes, teaches
       WHERE teacher.id = teaches.id AND student.id = takes.id AND
             takes.course_id = teaches.course_id AND
             takes.sec_id = teaches.sec_id AND
             takes.semester = teaches.semester AND
             takes.year = teaches.year
       GROUP BY teacher.name, student.name) AS T
WHERE T.totalcount >= 2
ORDER BY T.teachername;
```

5. Afficher les identifiants et les noms des étudiants qui n'ont pas suivi de cours en 2009.

```
SELECT student.id, student.name FROM student
EXCEPT
SELECT S.id, S.name FROM student S, takes
WHERE takes.id = S.id AND takes.year < 2009;
```

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

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

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

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

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

```
SELECT T1.name , T1.salary FROM teacher as T1
WHERE 2 >= (
  SELECT COUNT (DISTINCT T2.salary) FROM teacher as T2
  WHERE T2.salary < T1.salary
)
ORDER BY T1.salary ASC;
```

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

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

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

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

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

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

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

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

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

```
SELECT A.name, B.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.year = B.year
      AND A.id < B.id
GROUP BY A.id, B.id
HAVING COUNT(*) >= 1;
```

14. Afficher pour chaque enseignant, qui a effectivement assuré un cours, le nombre total d'étudiant 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.name, count(*)
FROM (takes INNER JOIN teaches USING (course_id, sec_id, semester, year))
INNER JOIN teacher ON teaches.id = teacher.id
GROUP BY teacher.name, teacher.id ORDER BY count(*) DESC;
```

15. Afficher pour chaque enseignant, même s'il n'a pas assuré de cours, le nombre total d'étudiant 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.name, count(course_id)
FROM (takes INNER JOIN teaches USING (course_id, sec_id, semester, year))
RIGHT OUTER JOIN teacher ON teaches.id = teacher.id
GROUP BY teacher.name, teacher.id ORDER BY count(course_id) DESC;
```

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

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

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

```
SELECT teacher.name, student.name, count(*)
FROM (teacher NATURAL JOIN teaches)
INNER JOIN takes
USING (course_id, sec_id, semester, year)
NATURAL JOIN student
GROUP BY teacher.name, student.name;
```

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

```
SELECT mytable.tn, mytable.sn
FROM (
  SELECT teacher.name AS tn, student.name AS sn, count(*)
  FROM (teacher NATURAL JOIN teaches)
  INNER JOIN takes
  NATURAL JOIN student
  USING (course_id, sec_id, semester, year)
  GROUP BY teacher.name, student.name
) AS mytable
WHERE count(*) >= 2;
```

Exercice 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 $F = \{A \to B; B \to C\}$
- Clé candidate : A (car $A \to B \to C \Rightarrow A \to B, C$)
- Dépendance transitive : $A \to C$ via B
- R est en 1NF (attributs atomiques)
- R est en 2NF (car dépendances concernent une seule clé)
- Mais pas en 3NF (car dépendance transitive $A \to C$)
- **Décomposition en 3NF** :
  - $R_1(A, B)$ avec $A \to B$
  - $R_2(B, C)$ avec $B \to C$

2. $R(A, B, C)$ et $F = \{A \to C; A \to B\}$
- Clé candidate : A (car $A \to B, C$)
- Toutes les dépendances sont directement dépendantes de la clé
- Pas de dépendance partielle ni transitive
- **Relation est en 3NF et BCNF**

3. $R(A, B, C)$ et $F = \{A, B \to C; C \to B\}$
- $A, B \to C \Rightarrow A, B$ est une clé candidate
- $C \to B$: dépendance partant d'un non-cle vers un attribut de cle $\Rightarrow$ pas en 3NF
- Pas en BCNF non plus (car $C \to B$ viole la définition)
- **Décomposition en BCNF** :
  - $R_1(C, B)$ avec $C \to B$
  - $R_2(A, C)$ car $A, B \to C \Rightarrow A, C \to B$

Exercice 3 :

1. Soit une relation $R(A, B, C, D, E)$ et un ensemble de dépendances fonctionnelles :

$$F = \{A \to B,\ C \to E,\ B \to D,\ E \to A\}$$

Déduire au moins 16 dépendances fonctionnelles en utilisant les règles d'Armstrong.

— À partir des dépendances données, on applique :
  — **Réflexivité** : $A \to A,\ B \to B, \ldots$
  — **Augmentation** : si $A \to B$, alors $AC \to BC$, etc.
  — **Transitivité** :

$$A \to B,\ B \to D \Rightarrow A \to D$$
$$E \to A,\ A \to B \Rightarrow E \to B$$
$$C \to E,\ E \to A \Rightarrow C \to A$$
$$C \to A,\ A \to B \Rightarrow C \to B$$
$$C \to B,\ B \to D \Rightarrow C \to D$$

— Parmi les dépendances dérivées :

$$A \to D,\ C \to A,\ C \to B,\ C \to D,\ E \to B,\ E \to D$$

2. Soit une relation $R(A, B, C, D, E, F)$ et :

$$F = \{A \to B,\ C \to D,\ B, C \to D,\ E \to B,\ D \to A\}$$

(a) **Calcul de la fermeture de** B^+ et $\{A, B\}^+$:

$$B^+ = \{B\} \quad \text{(aucune règle ne part de } B \text{ seul)}$$
$$\{A, B\}^+ = \{A, B\} \Rightarrow A \to B \Rightarrow \text{déjà inclus}$$

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

$$A \to B,\ B \to D,\ D \to A$$

(c) **Tester si la relation est en BCNF** :
  — $A \to B$: A n'est pas une super-clé ⇒ **pas en BCNF**
  — $E \to B$: E n'est pas super-clé ⇒ **pas en BCNF**
  Décomposition possible :

$$R_1(A, B), \quad R_2(E, B), \quad R_3(D, A), \quad R_4(C, D), \quad R_5(A, F)$$

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

(a) $R_1(A, B, C)$, $R_2(A, D, E)$:
  — Attribut commun : A
  — Si A est une clé dans au moins une des deux relations ⇒ **décomposition sans perte d'information**

(b) $R_1(A, B, C)$, $R_2(C, D, E)$:
  — Attribut commun : C
  — Si C n'est pas une clé dans aucune des deux relations ⇒ **perte d'information possible**

Exercice 4 :

1. Afficher la liste des dépendances :

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

2. Afficher la liste des relations :

```python
def printRelations(T: "list of relations"):
    for R in T:
        print("\t", R)
```

3. Retourner tous les sous-ensembles d'un ensemble donné :

```python
import itertools

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

4. Calcul de la fermeture d'un ensemble d'attributs K :

```python
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. Calcul de la fermeture de F (ensemble des dépendances déductibles) :

```python
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(R):
            if beta.issubset(computeAttributeClosure(F, K)):
                F_plus.append((K, beta))
    return F_plus
```

6. Tester si $\alpha \rightarrow \beta$ est une dépendance vraie :

```python
def isDependency(F, alpha: "set of attributes", beta: "set of attributes"):
    return beta.issubset(computeAttributeClosure(F, alpha))
```

7. Tester si K est une super-clé :

```
def isSuperKey(F, R, K):
    return R.issubset(computeAttributeClosure(F, K))
```

8. Tester si K est une clé candidate :

```
def isCandidateKey(F, R, K):
    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. Calculer toutes les clés candidates d'une relation :

```
def computeAllCandidateKeys(F, R):
    result = []
    for K in powerSet(R):
        if isCandidateKey(F, R, K):
            result.append(K)
    return result
```

10. Calculer toutes les super-clés :

```
def computeAllSuperKeys(F, R):
    result = []
    for K in powerSet(R):
        if isSuperKey(F, R, K):
            result.append(K)
    return result
```

11. Retourner une clé candidate :

```
def computeOneCandidateKey(F, R):
    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. Tester si une relation est en BCNF :

```
def isBCNFRelation(F, R):
    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. Tester si un schéma de relations est en BCNF :

```
def isBCNFRelations(F, T):
    for R in T:
        if isBCNFRelation(F, R)[0] == False:
            return False, R
    return True, {}
```

14. Implémenter la décomposition en BCNF :

```
def computeBCNFDecomposition(F, T):
    OUT, size = list(T), 0
    while size != len(OUT):
        size = len(OUT)
        for R in OUT:
            _is_BCNF, [alpha, beta] = isBCNFRelation(F, R)
            if not _is_BCNF:
                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
```