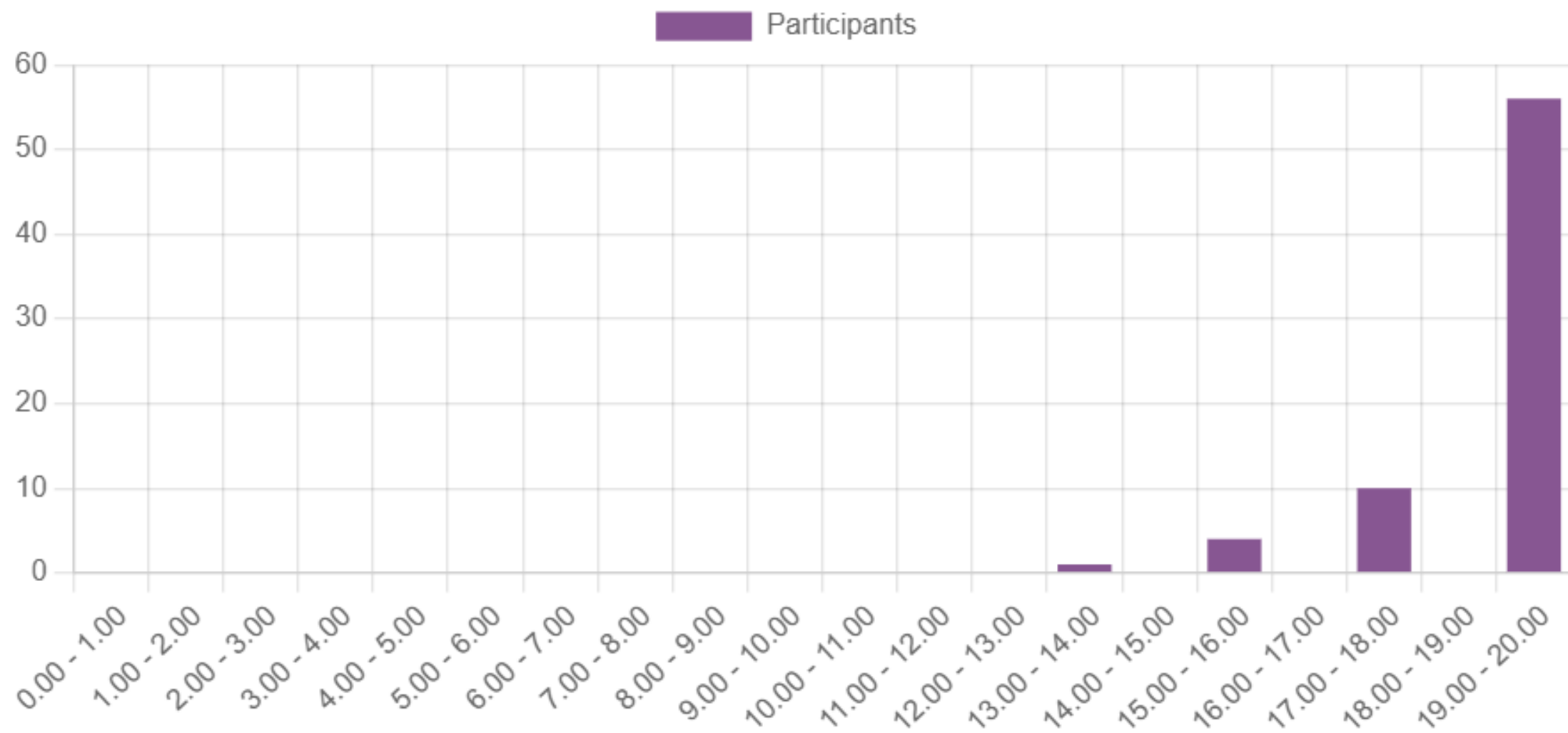


# TP Repas Agrégations - Quelques retours sur des « mauvaises pratiques »



### Question 1

Correct

Mark 1.00 out of 1.00

Flag question

**Donner le nombre total d'invités (différents)**

**Schéma attendu (nbInvites)**

Nous considérons le schéma suivant:

LesRepas (dateR, nomI)

LeMenu (dateR, nomP, nomV)

LesPreferences (nomA, nomP)

LesPlats (nomP, typeP)

Avec les contraintes d'intégrité référentielles suivantes:

LeMenu[dateR] = LesRepas[dateR]

LesPreferences[nomP]  $\subseteq$  LesPlats[nomP]

LeMenu[nomP]  $\subseteq$  LesPlats[nomP]

```
1 SELECT COUNT(*)
2 FROM (
3     SELECT DISTINCT nomI
4     FROM LesRepas);|
```



Requête mal  
construite ou trop  
compliquée



### Possible solution:

```
SELECT COUNT (DISTINCT nomI) AS nbInvites
FROM LesRepas;
```

## Question 2

Correct

Mark 1.00 out of 1.00

Flag question

### Donner la date du repas plus récent

#### Schéma attendu (datePlusRecent)

Nous considérons le schéma suivant:

LesRepas (dateR, nomI)

LeMenu (dateR, nomP, nomV)

LesPreferences (nomA, nomP)

LesPlats (nomP, typeP)

Avec les contraintes d'intégrité référentielles suivantes:

LeMenu[dateR] = LesRepas[dateR]

LesPreferences[nomP]  $\subseteq$  LesPlats[nomP]

LeMenu[nomP]  $\subseteq$  LesPlats[nomP]

```
1 SELECT DISTINCT dateR
2 FROM LesRepas R1
3 WHERE NOT EXISTS (
4     SELECT *
5     FROM LesRepas R2
6     WHERE R2.dateR > R1.dateR
7 );
8
```



```
1 WITH X AS (SELECT MAX(dateR) AS m
2             FROM LesRepas)
3 SELECT UNIQUE dateR
4 FROM LesRepas JOIN X ON (dateR = m);
```



#### Possible solution:

```
SELECT MAX(dateR) AS datePlusRecent
FROM LesRepas;
```

### Question 3

Correct

Mark 1.00 out of 1.00

Flag question

**Donner les noms des personnes qui ont été invitées au moins 2 fois (avec aggregation)**

**Schéma attendu (nomI)**

Nous considérons le schéma suivant:

LesRepas (dateR, nomI)

LeMenu (dateR, nomP, nomV)

LesPreferences (nomA, nomP)

LesPlats (nomP, typeP)

Avec les contraintes d'intégrité référentielles suivantes:

LeMenu[dateR] = LesRepas[dateR]

LesPreferences[nomP]  $\subseteq$  LesPlats[nomP]

LeMenu[nomP]  $\subseteq$  LesPlats[nomP]

```
1 WITH inviteParPersonnes AS (  
2     SELECT nomI, COUNT(dateR) AS nbDates  
3     FROM LesRepas  
4     GROUP BY nomI  
5 )  
6 SELECT nomI  
7 FROM inviteParPersonnes  
8 WHERE nbDates >= 2;
```



### Possible solution:

```
SELECT nomI  
FROM LesRepas  
GROUP BY nomI  
HAVING COUNT(dateR) >= 2;
```

**Question 4**

Correct

Mark 1.00 out of 1.00

Flag question

Donner le nombre moyen d'invités par repas arrondi à la centième (2 chiffres après le point). Pour cela, utiliser l'opérateur ROUND.

**Schéma attendu (nbMoy)**

Nous considérons le schéma suivant:

LesRepas (dateR, nomI)

LeMenu (dateR, nomP, nomV)

LesPreferences (nomA, nomP)

LesPlats (nomP, typeP)

Avec les contraintes d'intégrité référentielles suivantes:

LeMenu[dateR] = LesRepas[dateR]

LesPreferences[nomP] ⊆ LesPlats[nomP]

LeMenu[nomP] ⊆ LesPlats[nomP]

```
1 SELECT ROUND(AVG(nombreInvites), 2) AS nbMoy
2 FROM (
3     SELECT AVG (COUNT(DISTINCT nomI)) AS nombreInvites
4     FROM LesRepas
5     GROUP BY dateR);
6
```



```
1 SELECT ROUND(AVG(nb_i),2) FROM (SELECT dateR, COUNT(nomI) as nb_i
2 FROM LesRepas GROUP BY dateR);
```

**Possible solution:**

```
--Double aggregation pas possible sur SQLite (besoin d'une sous-requête).
SELECT ROUND(AVG(COUNT (nomI)),2) AS nbMoy
FROM LesRepas
GROUP BY dateR;
```

### Question 5

Correct

Mark 1.00 out of 1.00

Flag question

**Pour chaque invité, donner le nombre de repas auxquels il a été convié.**

**Schéma attendu (nomI, nbInvitations)**

Nous considérons le schéma suivant:

LesRepas (dateR, nomI)

LeMenu (dateR, nomP, nomV)

LesPreferences (nomA, nomP)

LesPlats (nomP, typeP)

Avec les contraintes d'intégrité référentielles suivantes:

LeMenu[dateR] = LesRepas[dateR]

LesPreferences[nomP]  $\subseteq$  LesPlats[nomP]

LeMenu[nomP]  $\subseteq$  LesPlats[nomP]

```
1 SELECT nomI as nomI, COUNT(dateR) as nbInvitations
2 FROM LesRepas GROUP BY nomI;
3
```



### Possible solution:

```
SELECT nomI, COUNT (dateR) as nbInvitations
FROM LesRepas
GROUP BY nomI;
```

**Question 6**

Correct

Mark 1.00 out of 1.00

Flag question

**Donner les noms des desserts qui ont été servis au moins 3 fois (avec partition)****Schéma attendu (nomP)**

Nous considérons le schéma suivant:

LesRepas (dateR, nomI)LeMenu (dateR, nomP, nomV)LesPreferences (nomA, nomP)LesPlats (nomP, typeP)

Avec les contraintes d'intégrité référentielles suivantes:

LeMenu[dateR] = LesRepas[dateR]

LesPreferences[nomP]  $\subseteq$  LesPlats[nomP]LeMenu[nomP]  $\subseteq$  LesPlats[nomP]

```
1 SELECT nomP
2 FROM LeMenu
3 JOIN LesPlats USING(nomP)
4 WHERE typeP='dessert'
5 GROUP BY nomP
6 HAVING COUNT(typeP)=3;
```



```
1 SELECT nomP
2 FROM LeMenu
3 GROUP BY nomP
4 HAVING COUNT (dateR) >= 3;
```



Requête fausse mais Caséine donne 20/20 car même résultat que la bonne réponse.

**Possible solution:**

```
SELECT nomP
FROM LeMenu JOIN LesPlats USING (nomP)
WHERE typeP = 'dessert'
GROUP BY nomP
HAVING COUNT(dateR) > 2;
```



```

1 WITH nonInvite AS( SELECT nomA, 0
2 FROM LesPreferences
3 MINUS
4 SELECT nomA, 0
5 FROM LesRepas JOIN LesPreferences ON (nomI = nomA))
6 SELECT nomA, COUNT (DISTINCT dateR) As nbInvitations
7 FROM LesRepas JOIN LesPreferences ON (nomI = nomA)
8 GROUP BY nomA
9 UNION
10 SELECT nomA, 0
11 FROM nonInvite;

```



```

SELECT nomA, COUNT (DISTINCT dateR) AS nbInvitations
FROM LesPreferences JOIN LesRepas ON (nomA=nomI)
GROUP BY nomA
UNION
(
SELECT nomA, 0
FROM LesPreferences
MINUS
SELECT nomI, 0
FROM LesRepas
);

```

**Question 8**

Correct

Mark 1.00 out of 1.00

Flag question

**Donner les noms des amis qui aiment tous les types de plats, c'est-à-dire au moins un plat de chaque type.**

**Schéma attendu (nomA)**

Nous considérons le schéma suivant:

LesRepas (dateR, nomI)

LeMenu (dateR, nomP, nomV)

LesPreferences (nomA, nomP)

LesPlats (nomP, typeP)


Avec les contraintes d'intégrité référentielles suivantes:

LeMenu[dateR] = LesRepas[dateR]


LesPreferences[nomP]  $\subseteq$  LesPlats[nomP]

LeMenu[nomP]  $\subseteq$  LesPlats[nomP]

```
1 SELECT nomA
2 FROM LesPreferences P
3 JOIN LesPlats M USING (nomP)
4 GROUP BY nomA
5 HAVING COUNT(DISTINCT typeP) = 9;
```



```
1 SELECT nomA
2 FROM LesPreferences
3 GROUP BY nomA
4 HAVING COUNT(nomP) = (SELECT COUNT(*)
5                       FROM LesPlats);
```

**Possible solution:**

```
--Possible avec WITH AS + JOIN
SELECT nomA
FROM LesPreferences JOIN LesPlats USING (nomP)
GROUP BY nomA
HAVING COUNT(DISTINCT typeP) IN
    (SELECT COUNT (DISTINCT typeP)
     FROM LesPlats);
```

**Question 9**

Correct

Mark 1.00 out of 1.00

Flag question

Pour l'(les) ami(s) invité(s) le plus souvent, donner son(leur) nom ainsi que ses(leurs) plats préférés.

**Schéma attendu (nomA, nomP)**

Nous considérons le schéma suivant:

LesRepas (dateR, nomI)

LeMenu (dateR, nomP, nomV)

LesPreferences (nomA, nomP)

LesPlats (nomP, typeP)

Avec les contraintes d'intégrité référentielles suivantes:

LeMenu[dateR] = LesRepas[dateR]

LesPreferences[nomP]  $\subseteq$  LesPlats[nomP]

LeMenu[nomP]  $\subseteq$  LesPlats[nomP]



```
1 With NbInvit AS (SELECT nomI, COUNT(dateR) as Nbinv
2 FROM LesRepas JOIN LesPreferences ON (nomI = nomA)
3 group by nomI),
4 MaxInvit AS (Select Max(Nbinv) as Maxinv from NbInvit),
5 PlusInv AS (Select nomI
6 From NbInvit Join MaxInvit on (Nbinv = Maxinv))
7 SELECT nomA, nomP
8 FROM LesPreferences JOIN PlusInv ON (nomA = nomI);
```

```
1 SELECT nomA, nomP
2 FROM LesPreferences
3 WHERE nomA = (
4     SELECT nomI
5     FROM LesRepas
6     GROUP BY (nomI)
7     HAVING COUNT(dateR) = 3
8     INTERSECT
9     SELECT nomA
10    FROM LesPreferences
11 );
```



### Possible solution:

```
--certains SGBD comme SQLite n'acceptent pas la double aggregation (MAX(COUNT(...))
WITH MaxInvitationsAmi AS (
    SELECT MAX(COUNT (DISTINCT dateR)) AS maxInvAmis
    FROM LesRepas JOIN LesPreferences ON (nomI=nomA)
    GROUP BY nomA),
InvitationsParAmi AS (
    SELECT nomA, COUNT (DISTINCT dateR) AS invAmis
    FROM LesRepas JOIN LesPreferences ON (nomI=nomA)
    GROUP BY nomA)
SELECT nomA, nomP
FROM LesPreferences JOIN InvitationsParAmi USING (nomA)
JOIN MaxInvitationsAmi ON (invAmis=maxInvAmis);
```

# Conclusion

## GENERALISER

Créer des requêtes génériques où le résultat ne dépend pas des données de la base de données, mais qui fonctionnent à chaque fois

## OPTIMISER

Eviter DISTINCT, JOINS, GROUP BY etc si on a pas besoin -> cela demande des ressources au système.

## SIMPLIFIER

Eviter d'utiliser des constructions trop complexes si on peut faire plus simple

## STYLISER

Attention au « style » des requêtes

- Homogène
- Sauts de ligne
- Simplicité