SQL : Langage d'Interrogation de Données

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Plan

Plan

MySQL

SQL

- Langage de définition de données
- Langage de manipulation de données
- Langage de contrôle de données
- **Langage d'interrogation de données**

MySQL

SQL

- Langage de définition de données
- Langage de manipulation de données
- Langage de contrôle de données
- **Langage d'interrogation de données**

Langage d'interrogation de données

- Langage qui permet de lire de données stockées dans la base de données
- Utilisant des concepts connus en algèbre relationnel : projection, jointure, intersection, union, produit cartésien...

Considérons la base de données `cours_lid` contenant les deux tables suivantes

personnes				
id	nom	prenom	salaire	ville
1	Cohen	Sophie	2000	Marseille
2	Leberre	Bernard	1500	Marseille
3	Benamar	Pierre	1800	Lyon
4	Hadad	Karim	2500	Paris
5	Wick	John	3000	Paris

`id` : clé primaire

vehicules				
immatriculation	marque	modele	annee	id_personne
100	Peugeot	5008	2018	5
200	Renault	clio	2000	4
300	Ford	fiesta	2010	1
400	Peugeot	106	2002	3
500	Citroen	C4	2015	4
600	Ford	Kuga	2019	
700	Fiat	punto	2008	5

`immatriculation` : clé primaire

`id_personne` : clé étrangère

MySQL

Script de la création de la base de données

```
CREATE DATABASE cours_lid;

use cours_lid;

CREATE TABLE personnes (
  id INT PRIMARY KEY,
  nom varchar(20),
  prenom varchar(20),
  salaire int(4),
  ville varchar(20)
)ENGINE=InnoDB;

CREATE TABLE vehicules (
  immatriculation INT PRIMARY KEY,
  marque varchar(20),
  modele varchar(20),
  annee int(4),
  id_personne int(3),
  CONSTRAINT fk_vehicules_personnes FOREIGN KEY (id_personne)
    REFERENCES personnes(id)
)ENGINE=InnoDB;
```

MySQL

Script d'insertion de données

```
INSERT INTO personnes VALUES
    (1, 'Cohen', 'Sophie', 2000, 'Marseille'),
    (2, 'Leberre', 'Bernard',  1500, 'Marseille'),
    (3, 'Benamar', 'Pierre',  1800, 'Lyon'),
    (4, 'Hadad', 'Karim',  2500, 'Paris'),
    (5, 'Wick', 'John',  3000, 'Paris');

INSERT INTO vehicules VALUES
    (100, 'Peugeot', '5008', 2018, 5),
    (200, 'Renault', 'clio',  2000, 4),
    (300, 'Ford', 'fiesta',  2010, 1),
    (400, 'Peugeot', '106',  2002, 3),
    (500, 'Citroen', 'C4',  2015, 4),
    (600, 'Ford', 'Kuga',  2019, null),
    (700, 'Fiat', 'punto',  2008, 5);
```

MySQL

Remarque

Une requête **SQL** de lecture est composée de deux clauses obligatoires : `SELECT` et `FROM`.

MySQL

Remarque

Une requête **SQL** de lecture est composée de deux clauses obligatoires : `SELECT` et `FROM`.

Structure d'une requête SQL

```
SELECT nom_colonne(s)
FROM nom_table(s)
WHERE condition(s)
GROUP BY nom_colonne(s)
HAVING condition(s)
ORDER BY nom_colonne(s)
LIMIT nombre
OFFSET nombre;
```

MySQL

Pour sélectionner toutes les données de la table `personnes`

```
SELECT *
FROM  personnes;
```

MySQL

Pour sélectionner toutes les données de la table `personnes`

```
SELECT *
FROM  personnes;
```

Le résultat

```
+-----+----------+----------+----------+------------+
| id  | nom      | prenom   | salaire  | ville      |
+-----+----------+----------+----------+------------+
|   1 | Cohen    | Sophie   |     2000 | Marseille  |
|   2 | Leberre  | Bernard  |     1500 | Marseille  |
|   3 | Benamar  | Pierre   |     1800 | Lyon       |
|   4 | Hadad    | Karim    |     2500 | Paris      |
|   5 | Wick     | John     |     3000 | Paris      |
+-----+----------+----------+----------+------------+
```

MySQL

Et si on veut sélectionner que les deux premières personnes

```
SELECT *
FROM  personnes
LIMIT 2;
```

MySQL

Et si on veut sélectionner que les deux premières personnes

```
SELECT *
FROM  personnes
LIMIT 2;
```

Le résultat

```
+-----+----------+----------+----------+------------+
| id  | nom      | prenom   | salaire  | ville      |
+-----+----------+----------+----------+------------+
|   1 | Cohen    | Sophie   |     2000 | Marseille  |
|   2 | Leberre  | Bernard  |     1500 | Marseille  |
+-----+----------+----------+----------+------------+
```

MySQL

Et si on veut sélectionner les deux personnes suivantes

```
SELECT *
FROM  personnes
LIMIT 2
OFFSET 2;
```

MySQL

Et si on veut sélectionner les deux personnes suivantes

```
SELECT *
FROM  personnes
LIMIT 2
OFFSET 2;
```

Le résultat

```
+-----+---------+--------+---------+-------+
| id  | nom     | prenom | salaire | ville |
+-----+---------+--------+---------+-------+
|   3 | Benamar | Pierre |    1800 | Lyon  |
|   4 | Hadad   | Karim  |    2500 | Paris |
+-----+---------+--------+---------+-------+
```

MySQL

On peut aussi simplifier l'écriture précédente

```
SELECT *
FROM  personnes
LIMIT 2, 2;
```

MySQL

On peut aussi simplifier l'écriture précédente

```
SELECT *
FROM  personnes
LIMIT 2, 2;
```

Le résultat est le même

```
+-----+---------+--------+---------+-------+
| id  | nom     | prenom | salaire | ville |
+-----+---------+--------+---------+-------+
|   3 | Benamar | Pierre |    1800 | Lyon  |
|   4 | Hadad   | Karim  |    2500 | Paris |
+-----+---------+--------+---------+-------+
```

MySQL

Remarques

- Les clauses `OFFSET` et `LIMIT` ne font pas partie du **SQL-ANSI**.
- Elles sont spécifiques à **MySQL**, **PostgreSQL**, **SQLite** et **MariaDB**.
- Certains autres, comme **SQL Server**, utilisent `TOP`.
- Le standard **SQL-ANSI** (depuis **SQL-2008**) utilise les clauses :
  - `FETCH FIRST n ROWS ONLY` (Limite les lignes retournées)
  - `OFFSET n ROWS FETCH NEXT m ROWS ONLY` (Pagination avec décalage et limite)

MySQL

Pour filtrer les colonnes (ici la ville de chaque personne)

```
SELECT ville
FROM  personnes;
```

Le résultat

```
+-----------+
| ville     |
+-----------+
| Marseille |
| Marseille |
| Lyon      |
| Paris     |
| Paris     |
+-----------+
```

MySQL

Pour filtrer les colonnes (ici la ville de chaque personne)

```
SELECT ville
FROM  personnes;
```

Le résultat

```
+-----------+
| ville     |
+-----------+
| Marseille |
| Marseille |
| Lyon      |
| Paris     |
| Paris     |
+-----------+
```

Comment faire pour supprimer les doublons ?

MySQL

Pour supprimer les doublons

```
SELECT distinct(ville)
FROM  personnes;
```

Le résultat

```
+-----------+
| ville     |
+-----------+
| Marseille |
| Lyon      |
| Paris     |
+-----------+
```

Pour compter le nombre de ville dans la table `personnes`

```
SELECT COUNT(distinct(ville))
FROM  personnes;
```

Le résultat

```
+-------------------------+
| COUNT(distinct(ville)) |
+-------------------------+
|                       3 |
+-------------------------+
```

Pour compter le nombre de ville dans la table `personnes`

```
SELECT COUNT(distinct(ville))
FROM  personnes;
```

Le résultat

```
+-------------------------+
| COUNT(distinct(ville)) |
+-------------------------+
|                       3 |
+-------------------------+
```

Remarque

`count` **compte les tuples (les lignes) et pas les colonnes.**

Pour compter le nombre de ville dans la table `personnes`

```
SELECT COUNT(distinct(ville))
FROM  personnes;
```

Le résultat

```
+-------------------------+
| COUNT(distinct(ville)) |
+-------------------------+
|                       3 |
+-------------------------+
```

Remarque

`count` **compte les tuples (les lignes) et pas les colonnes.**

Question

Comment remplacer le nom de la colonne `COUNT(distinct(ville))` **par un autre personnalisé ?**

MySQL

Pour modifier le nom d'une colonne dans l'affichage du résultat, on peut utiliser les alias

```
SELECT COUNT(distinct(ville)) as nombre_ville
FROM  personnes;
```

Le résultat

```
+--------------+
| nombre_ville |
+--------------+
|            3 |
+--------------+
```

MySQL

Pour avoir un alias contenant un espace

```
SELECT COUNT(distinct(ville)) as "nombre ville"
FROM  personnes;
```

Le résultat

```
+--------------+
| nombre ville |
+--------------+
|            3 |
+--------------+
```

MySQL

Remarque

- Les alias peuvent porter sur les colonnes mais aussi sur les tables (à voir dans la section sur les jointures).
- Pour les alias contenant des espaces, certains **SGBD** comme **MS SQL Server**, **Sybase**, et **Azure SQL Database** utilisent les `[alias avec espace]`.
- D'autres SGBD, comme **MySQL**, **PostgreSQL**, **Oracle** et **MariaDB**, utilisent des guillemets doubles `"alias avec espace"` selon la norme **SQL-ANSI**.

MySQL

Pour sélectionner les personnes dont le salaire = 2000

```
SELECT *
FROM  personnes
WHERE  salaire = 2000;
```

Le résultat

```
+-----+----------+---------+----------+------------+
| id  | nom      | prenom  | salaire | ville      |
+-----+----------+---------+----------+------------+
|   1 | Cohen    | Sophie  |    2000 | Marseille |
+-----+----------+---------+----------+------------+
```

MySQL

Opérateurs de comparaison

	MySQL	SQL-ANSI
= (égalité entre valeurs non NULL)	✓	✓
!=	✓	✓
<>	✓	✓
<	✓	✓
<=	✓	✓
>	✓	✓
>=	✓	✓

MySQL

Autres opérateurs de comparaison

	MySQL	SQL-ANSI
`BETWEEN`	✓	✓
`IN`	✓	✓
`NOT IN`	✓	✓
`IS NULL`	✓	✓
`IS NOT NULL`	✓	✓
`<=>` (égalité même avec les valeurs `NULL`)	✓	⊘

MySQL

Exercice 1

Écrire une requête **SQL** qui permet de sélectionner les personnes qui habitent à `Marseille`.

MySQL

Pour sélectionner les personnes dont la ville = 'Marseille'

```
SELECT *
FROM  personnes
WHERE   ville = 'Marseille';
```

Le résultat

```
+-----+---------+---------+---------+-----------+
| id  | nom     | prenom  | salaire | ville     |
+-----+---------+---------+---------+-----------+
|   1 | Cohen   | Sophie  |    2000 | Marseille |
|   2 | Leberre | Bernard |    1500 | Marseille |
+-----+---------+---------+---------+-----------+
```

MySQL

Remarque

- Par défaut, les requêtes de recherche sont insensibles à la casse en **MySQL**
- Par conséquence, `WHERE ville = 'Marseille'`, `WHERE ville = 'MARSEILLE'` et `WHERE ville = 'marseille'` retournent toutes le même résultat.

MySQL

Pour sélectionner les personnes dont la ville = 'Marseille' en faisant attention à la casse

```
SELECT *
FROM  personne
WHERE  BINARY ville = 'Marseille';
```

Le résultat

```
+-----+----------+----------+----------+-----------+
| id  | nom      | prenom   | salaire | ville      |
+-----+----------+----------+----------+-----------+
|   1 | Cohen    | Sophie   |    2000 | Marseille |
|   2 | Leberre  | Bernard  |    1500 | Marseille |
+-----+----------+----------+----------+-----------+
```

MySQL

Ou

```
SELECT *
FROM  personnes
WHERE ville = BINARY 'Marseille';
```

Le résultat

```
+-----+----------+---------+----------+-----------+
| id  | nom      | prenom  | salaire | ville      |
+-----+----------+---------+----------+-----------+
|   1 | Cohen    | Sophie  |     2000 | Marseille |
|   2 | Leberre | Bernard |     1500 | Marseille |
+-----+----------+---------+----------+-----------+
```

MySQL

Remarque

Il est possible d'enchaîner les conditions en utilisant les opérateurs logiques.

MySQL

	MySQL	SQL-ANSI
AND	✓	✓
OR	✓	✓
NOT	✓	✓
XOR	✓	⊘
&& (alias de AND)	✓	⊘
! (alias de NOT)	✓	⊘

MySQL

Exercice 2

Écrire une requête **SQL** qui permet de sélectionner les personnes qui habitent à `Marseille` ou à `Lyon`.

MySQL

Exercice 2

Écrire une requête **SQL** qui permet de sélectionner les personnes qui habitent à `Marseille` ou à `Lyon`.

Exercice 3

Écrire une requête **SQL** qui permet de sélectionner les personnes dont le salaire est compris entre `2000` et `3000`.

MySQL

Pour l'exercice 2, on peut aussi utiliser `between` et `and`

```
SELECT *
FROM  personnes
WHERE   salaire BETWEEN  2000 AND 3000;
```

Le résultat

```
+-----+-------+--------+---------+-----------+
| id  | nom   | prenom | salaire | ville     |
+-----+-------+--------+---------+-----------+
|   1 | Cohen | Sophie |    2000 | Marseille |
|   4 | Hadad | Karim  |    2500 | Paris     |
|   5 | Wick  | John   |    3000 | Paris     |
+-----+-------+--------+---------+-----------+
```

MySQL

Pour sélectionner les personnes ayant un salaire = 2000 ou 3000

```
SELECT *
FROM  personnes
WHERE  salaire IN (2000, 3000);
```

Le résultat

```
+-----+-------+--------+---------+-----------+
| id  | nom   | prenom | salaire | ville     |
+-----+-------+--------+---------+-----------+
|   1 | Cohen | Sophie |    2000 | Marseille |
|   5 | Wick  | John   |    3000 | Paris     |
+-----+-------+--------+---------+-----------+
```

MySQL

Exercice 4

Écrire une requête **SQL** qui permet de sélectionner les personnes qui habitent Marseille et dont le salaire est soit inférieur à 2000 soit supérieur à 2500.

MySQL

Pour l'exercice 3, on peut aussi utiliser `not` et `between`

```
SELECT *
FROM  personnes
WHERE  ville = 'Marseille'
AND salaire NOT BETWEEN  2000 AND 2500;
```

Le résultat

```
+-----+----------+----------+----------+-----------+
| id  | nom      | prenom   | salaire | ville      |
+-----+----------+----------+----------+-----------+
|   2 | Leberre | Bernard |    1500 | Marseille |
+-----+----------+----------+----------+-----------+
```

MySQL

Pour sélectionner les personnes dont le nom de la ville contient le caractère `a`

```
SELECT *
FROM  personnes
WHERE  ville like '%a%';
```

Le résultat

```
+-----+---------+---------+---------+-----------+
| id  | nom     | prenom  | salaire | ville     |
+-----+---------+---------+---------+-----------+
|   1 | Cohen   | Sophie  |    2000 | Marseille |
|   2 | Leberre | Bernard |    1500 | Marseille |
|   4 | Hadad   | Karim   |    2500 | Paris     |
|   5 | Wick    | John    |    3000 | Paris     |
+-----+---------+---------+---------+-----------+
```

MySQL

Caractères génériques de l'opérateur `Like`

- `%` : désigne 0, 1 ou plusieurs caractères.
- `_` : désigne un seul caractère.
- `[]` : vérifie la présence d'un caractère de la liste, non-supporté par **MySQL**.
- `^` : vérifie l'absence de tous les caractères de la liste, non-supporté par **MySQL**.
- `–` : définit un intervalle de valeur, non-supporté par **MySQL**.

MySQL

Pour sélectionner les véhicules dont le numéro du propriétaire est nul

```
SELECT *
FROM  vehicules
WHERE  id_personne IS NULL;
```

Le résultat

```
+-----------------+--------+--------+-------+--------------+
| immatriculation | marque | modele | annee | id_personne  |
+-----------------+--------+--------+-------+--------------+
|             600 | Ford   | Kuga   |  2019 |         NULL |
+-----------------+--------+--------+-------+--------------+
```

MySQL

En MySQL, on peut également utiliser l'opérateur <=> pour faire la même chose

```
SELECT *
FROM  vehicules
WHERE  id_personne <=> NULL;
```

Le résultat

```
+-----------------+--------+--------+-------+--------------+
| immatriculation | marque | modele | annee | id_personne |
+-----------------+--------+--------+-------+--------------+
|             600 | Ford   | Kuga   |  2019 |         NULL |
+-----------------+--------+--------+-------+--------------+
```

MySQL

Attention, utiliser = NULL ne retourne pas TRUE

```
SELECT *
FROM  vehicules
WHERE  id_personne = NULL;
```

Le résultat

```
Empty set (0.00 sec)
```

MySQL

Pour remplacer les valeurs nulles par une autre valeur, on peut utiliser la fonction `COALESCE()`

```
SELECT marque, COALESCE(id_personne, 'sans propriétaire') AS proprié
   taire
FROM  vehicules;
```

Le résultat

```
+--------+------------------+
| marque | propriétaire     |
+--------+------------------+
| Peugeot | 5               |
| Renault | 4               |
| Ford    | 1               |
| Peugeot | 3               |
| Citroen | 4               |
| Ford    | sans propriétaire |
| Fiat    | 5               |
+--------+------------------+
```

MySQL

`COALESCE` est conforme à la norme SQL-ANSI et est pris en charge par la plupart des SGBD, `IFNULL` est spécifique à MySQL et MariaDB

```
SELECT marque, IFNULL(id_personne, 'sans propriétaire') AS propriétaire
FROM  vehicules;
```


Le résultat

```
+---------+--------------------+
| marque  | propriétaire       |
+---------+--------------------+
| Peugeot | 5                  |
| Renault | 4                  |
| Ford    | 1                  |
| Peugeot | 3                  |
| Citroen | 4                  |
| Ford    | sans propriétaire  |
| Fiat    | 5                  |
+---------+--------------------+
```

MySQL

Pour ordonner le résultat selon le numéro du propriétaire

```
SELECT *
FROM  vehicules
WHERE  id_personne IS NOT NULL
ORDER BY  id_personne;
```

Le résultat

```
+-----------------+---------+--------+-------+-------------+
| immatriculation | marque  | modele | annee | id_personne |
+-----------------+---------+--------+-------+-------------+
|             300 | Ford    | fiesta |  2010 |           1 |
|             400 | Peugeot | 106    |  2002 |           3 |
|             200 | Renault | clio   |  2000 |           4 |
|             500 | Citroen | C4     |  2015 |           4 |
|             100 | Peugeot | 5008   |  2018 |           5 |
|             700 | Fiat    | punto  |  2008 |           5 |
+-----------------+---------+--------+-------+-------------+
```

MySQL

Deux options possibles pour `ORDER BY`

- `ASC` (par défaut)
- `DESC`

MySQL

Pour ordonner le résultat selon deux critères ou plus

```
SELECT *
FROM  vehicules
WHERE  id_personne IS NOT NULL
ORDER BY  id_personne, annee DESC;
```

Le résultat

immatriculation	marque	modele	annee	id_personne
300	Ford	fiesta	2010	1
400	Peugeot	106	2002	3
500	Citroen	C4	2015	4
200	Renault	clio	2000	4
100	Peugeot	5008	2018	5
700	Fiat	punto	2008	5

MySQL

Pour regrouper les données de la table personne par `ville` et calculer le salaire total dans chaque ville

```
SELECT ville, SUM(salaire) AS total_salaire
FROM personnes
GROUP BY ville;
```

Le résultat

```
+-----------+----------------+
| ville     | total_salaire  |
+-----------+----------------+
| Marseille |           3500 |
| Lyon      |           1800 |
| Paris     |           5500 |
+-----------+----------------+
```

MySQL

Exercice 5

Écrire une requête **SQL** qui permet de compter le nombre de véhicule pour chaque personne.

Le résultat attendu

```
+-------------+------------------+
| id_personne | nombre_vehicule  |
+-------------+------------------+
|           1 |                1 |
|           3 |                1 |
|           4 |                2 |
|           5 |                2 |
+-------------+------------------+
```

Solution

```
SELECT id_personne, COUNT(*) AS nombre_vehicule
FROM  vehicules
WHERE  id_personne is not null
GROUP BY id_personne;
```

MySQL

Pour regrouper les données de la table personne par ville et calculer le salaire total dans chaque ville, avec une ligne supplémentaire pour afficher le total

```
SELECT ville, SUM(salaire) AS total_salaire
FROM personnes
GROUP BY ville WITH ROLLUP;
```

Le résultat

```
+-----------+----------------+
| ville     | total_salaire  |
+-----------+----------------+
| Lyon      |           1800 |
| Marseille |           3500 |
| Paris     |           5500 |
| NULL      |          10800 |
+-----------+----------------+
```

MySQL

On peut utiliser `IFNULL()` ou `COALESCE()` pour remplacer `NULL` par une étiquette comme `Total général`

```
SELECT
    IFNULL(ville, 'Total général') AS ville,
    SUM(salaire) AS total_salaire
FROM personnes
GROUP BY ville WITH ROLLUP;
```

Le résultat

```
+---------------+---------------+
| ville         | total_salaire |
+---------------+---------------+
| Lyon          |          1800 |
| Marseille     |          3500 |
| Paris         |          5500 |
| Total général |         10800 |
+---------------+---------------+
```

MySQL

Pour filtrer les résultats de la fonction d'agrégation

```
SELECT id_personne, COUNT(*) as nombre_vehicule
FROM  vehicules
WHERE   id_personne IS NOT NULL
GROUP BY id_personne
HAVING   nombre_vehicule > 1;
```

Le résultat

```
+-------------+-----------------+
| id_personne | nombre_vehicule |
+-------------+-----------------+
|           4 |               2 |
|           5 |               2 |
+-------------+-----------------+
```

MySQL

Fonctions d'agrégation

- MAX
- MIN
- COUNT
- SUM
- AVG

MySQL

Remarques

- Imbriquer les fonctions d'agrégation n'est pas possible.
- `distinct` n'est pas une fonction d'agrégation.
- Pas d'espace entre la fonction d'agrégation et la parenthèse ouvrante (par exemple : `MAX(...)`)

MySQL

La fonction d'agrégation peut ne pas apparaître dans le SELECT

```
SELECT id_personne
FROM  vehicules
WHERE   id_personne IS NOT NULL
GROUP BY id_personne
HAVING   COUNT(*) > 1;
```

Le résultat

```
+-------------+
| id_personne |
+-------------+
|           4 |
|           5 |
+-------------+
```

MySQL

Exercice 6

Écrire une requête **SQL** qui permet de compter la somme des salaires par ville.

MySQL

Exercice 6

Écrire une requête **SQL** qui permet de compter la somme des salaires par ville.

Exercice 7

Écrire une requête **SQL** qui permet de sélectionner les numéros de personne qui ont un véhicule de marque `Renault` ou `Citroen`.

MySQL

Opérateurs ensemblistes

- `UNION`
- `INTERSECT`
- `EXCEPT`

MySQL

Pour répondre à la question de l'exercice 5, on peut utiliser `UNION`

```
SELECT id_personne
FROM  vehicules
WHERE   marque = 'Renault'
UNION
SELECT id_personne
FROM  vehicules
WHERE   marque = 'Citroen';
```


Le résultat

```
+-------------+
| id_personne |
+-------------+
|           4 |
+-------------+
```

MySQL

Pour afficher les doublons, on peut utiliser `UNION ALL`

```
SELECT id_personne
FROM  vehicules
WHERE   marque = 'Renault'
UNION ALL
SELECT id_personne
FROM  vehicules
WHERE   marque = 'Citroen';
```

Le résultat

```
+-------------+
| id_personne |
+-------------+
|           4 |
|           4 |
+-------------+
```

MySQL

Exercice 8

Écrire une requête **SQL** qui permet de sélectionner les numéros de personne qui ont un véhicule de marque `Fiat` et un de marque `Peugeot`.

MySQL

Pour répondre à la question de l'exercice précédent, on peut utiliser `INTERSECT` [Dispoible depuis MySQL 8.0.31]

```
SELECT id_personne
FROM  vehicules
WHERE   marque = 'Fiat'
INTERSECT
SELECT id_personne
FROM  vehicules
WHERE   marque = 'Peugeot';
```

Le résultat

```
+-------------+
| id_personne |
+-------------+
|           5 |
+-------------+
```

MySQL

Exercice 9

Écrire une requête **SQL** qui permet de sélectionner les numéros de personne qui ont un véhicule de marque `Peugeot` mais pas un de marque `Fiat`.

MySQL

Pour répondre à la question de l'exercice précédent, on peut utiliser `EXCEPT` [Dispoible depuis MySQL 8.0.31]

```
SELECT id_personne
FROM  vehicules
WHERE   marque = 'Peugeot'
EXCEPT
SELECT id_personne
FROM  vehicules
WHERE   marque = 'Fiat';
```


Le résultat

```
+-------------+
| id_personne |
+-------------+
|           3 |
+-------------+
```

MySQL

Remarques

- Pour les propriétaires des véhicules, on affichait chaque fois le numéro.
- Pour l'utilisateur, il serait mieux de connaître les nom et prénom que le numéro.
- Les nom et prénom sont dans la table personne.

MySQL

Remarques

- Pour les propriétaires des véhicules, on affichait chaque fois le numéro.
- Pour l'utilisateur, il serait mieux de connaître les nom et prénom que le numéro.
- Les nom et prénom sont dans la table personne.

Solution

Les jointures

MySQL

Deux types de jointure

- Implicite : sans le mot-clé `join`
- Explicite : avec le mot-clé `join`

MySQL

Pour sélectionner toutes les données de la table `personnes`

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes, vehicules
WHERE   personnes.id = vehicules.id_personne;
```

Le résultat

```
+--------+--------+-----------+---------+--------+
| nom    | prenom | ville     | marque  | modele |
+--------+--------+-----------+---------+--------+
| Cohen  | Sophie | Marseille | Ford    | fiesta |
| Benamar| Pierre | Lyon      | Peugeot | 106    |
| Hadad  | Karim  | Paris     | Renault | clio   |
| Hadad  | Karim  | Paris     | Citroen | C4     |
| Wick   | John   | Paris     | Peugeot | 5008   |
| Wick   | John   | Paris     | Fiat    | punto  |
+--------+--------+-----------+---------+--------+
```

MySQL

Remarques

- La jointure se fait, généralement, sur une colonne commune entre deux tables (clé primaire dans une première table qui est étrangère dans une deuxième).
- En cas d'ambiguïté, c'est-à-dire, si deux colonnes portent le même nom, il faut les préfixer par le nom de leurs tables respectives.
- Il est aussi possible de définir et d'utiliser des alias.
- Les personnes n'ont pas de véhicules et les véhicules sont propriétaires n'apparaissent pas dans le résultat.

MySQL

On peut aussi utiliser les alias

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes p, vehicules v
WHERE   p.id = v.id_personne;
```


Le résultat est le même

```
+---------+--------+-----------+---------+--------+
| nom     | prenom | ville     | marque  | modele |
+---------+--------+-----------+---------+--------+
| Cohen   | Sophie | Marseille | Ford    | fiesta |
| Benamar | Pierre | Lyon      | Peugeot | 106    |
| Hadad   | Karim  | Paris     | Renault | clio   |
| Hadad   | Karim  | Paris     | Citroen | C4     |
| Wick    | John   | Paris     | Peugeot | 5008   |
| Wick    | John   | Paris     | Fiat    | punto  |
+---------+--------+-----------+---------+--------+
```

MySQL

En l'absence d'ambiguïté, on peut aussi écrire

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes, vehicules
WHERE   id = id_personne;
```

Le résultat est le même

```
+--------+--------+-----------+---------+--------+
| nom    | prenom | ville     | marque  | modele |
+--------+--------+-----------+---------+--------+
| Cohen  | Sophie | Marseille | Ford    | fiesta |
| Benamar| Pierre | Lyon      | Peugeot | 106    |
| Hadad  | Karim  | Paris     | Renault | clio   |
| Hadad  | Karim  | Paris     | Citroen | C4     |
| Wick   | John   | Paris     | Peugeot | 5008   |
| Wick   | John   | Paris     | Fiat    | punto  |
+--------+--------+-----------+---------+--------+
```

MySQL

Jointure explicite : plusieurs syntaxes [**SQL 2**]

- `INNER JOIN ... ON` : exactement comme la jointure implicite.
- `LEFT JOIN ... ON` : jointure gauche.
- `RIGHT JOIN ... ON` : jointure droite.
- `CROSS JOIN ... ON` : jointure interne + droite + gauche.
- `FULL JOIN ... ON` : **SQL-ANSI** non supporté par **MySQL**, mais on peut le simuler avec `UNION` de `LEFT JOIN` et `RIGHT JOIN`.
- ...

MySQL

On peut utiliser le mot-clé `INNER JOIN` et indiquer les colonnes sur lesquelles on fait la jointure

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes
INNER JOIN vehicules ON id = id_personne;
```

Le résultat est le même

```
+---------+--------+-----------+---------+--------+
| nom     | prenom | ville     | marque  | modele |
+---------+--------+-----------+---------+--------+
| Cohen   | Sophie | Marseille | Ford    | fiesta |
| Benamar | Pierre | Lyon      | Peugeot | 106    |
| Hadad   | Karim  | Paris     | Renault | clio   |
| Hadad   | Karim  | Paris     | Citroen | C4     |
| Wick    | John   | Paris     | Peugeot | 5008   |
| Wick    | John   | Paris     | Fiat    | punto  |
+---------+--------+-----------+---------+--------+
```

MySQL

`JOIN` **est le raccourci de** `INNER JOIN`

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes
JOIN vehicules ON id = id_personne;
```


Le résultat est le même

```
+---------+--------+-----------+---------+--------+
| nom     | prenom | ville     | marque  | modele |
+---------+--------+-----------+---------+--------+
| Cohen   | Sophie | Marseille | Ford    | fiesta |
| Benamar | Pierre | Lyon      | Peugeot | 106    |
| Hadad   | Karim  | Paris     | Renault | clio   |
| Hadad   | Karim  | Paris     | Citroen | C4     |
| Wick    | John   | Paris     | Peugeot | 5008   |
| Wick    | John   | Paris     | Fiat    | punto  |
+---------+--------+-----------+---------+--------+
```

MySQL

Pour afficher aussi les personnes qui n'ont pas de voiture, on peut écrire

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes
LEFT JOIN vehicules ON id = id_personne;
```


Le résultat

```
+--------+--------+----------+--------+--------+
| nom    | prenom | ville    | marque | modele |
+--------+--------+----------+--------+--------+
| Cohen  | Sophie | Marseille| Ford   | fiesta |
| Leberre| Bernard| Marseille| NULL   | NULL   |
| Benamar| Pierre | Lyon     | Peugeot| 106    |
| Hadad  | Karim  | Paris    | Renault| clio   |
| Hadad  | Karim  | Paris    | Citroen| C4     |
| Wick   | John   | Paris    | Peugeot| 5008   |
| Wick   | John   | Paris    | Fiat   | punto  |
+--------+--------+----------+--------+--------+
```

MySQL

Le raccourci de `LEFT OUTER JOIN ... ON` **est** `LEFT JOIN ... ON`

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes
LEFT JOIN vehicules ON id = id_personne;
```


Le résultat

```
+---------+---------+-----------+---------+--------+
| nom     | prenom  | ville     | marque  | modele |
+---------+---------+-----------+---------+--------+
| Cohen   | Sophie  | Marseille | Ford    | fiesta |
| Leberre | Bernard | Marseille | NULL    | NULL   |
| Benamar | Pierre  | Lyon      | Peugeot | 106    |
| Hadad   | Karim   | Paris     | Renault | clio   |
| Hadad   | Karim   | Paris     | Citroen | C4     |
| Wick    | John    | Paris     | Peugeot | 5008   |
| Wick    | John    | Paris     | Fiat    | punto  |
+---------+---------+-----------+---------+--------+
```

MySQL

Pour afficher les véhicules qui n'ont pas de propriétaire, on peut écrire

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes
RIGHT OUTER JOIN vehicules ON id = id_personne;
```

Le résultat

```
+---------+--------+-----------+---------+--------+
| nom     | prenom | ville     | marque  | modele |
+---------+--------+-----------+---------+--------+
| Cohen   | Sophie | Marseille | Ford    | fiesta |
| Benamar | Pierre | Lyon      | Peugeot | 106    |
| Hadad   | Karim  | Paris     | Renault | clio   |
| Hadad   | Karim  | Paris     | Citroen | C4     |
| Wick    | John   | Paris     | Peugeot | 5008   |
| Wick    | John   | Paris     | Fiat    | punto  |
| NULL    | NULL   | NULL      | Ford    | Kuga   |
+---------+--------+-----------+---------+--------+
```

MySQL

Le raccourci de `RIGHT OUTER JOIN ... ON` **est** `RIGHT JOIN ... ON`

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes
RIGHT JOIN vehicules ON id = id_personne;
```

Le résultat

```
+---------+--------+-----------+---------+--------+
| nom     | prenom | ville     | marque  | modele |
+---------+--------+-----------+---------+--------+
| Cohen   | Sophie | Marseille | Ford    | fiesta |
| Benamar | Pierre | Lyon      | Peugeot | 106    |
| Hadad   | Karim  | Paris     | Renault | clio   |
| Hadad   | Karim  | Paris     | Citroen | C4     |
| Wick    | John   | Paris     | Peugeot | 5008   |
| Wick    | John   | Paris     | Fiat    | punto  |
| NULL    | NULL   | NULL      | Ford    | Kuga   |
+---------+--------+-----------+---------+--------+
```

MySQL

Pour le produit cartésien, on utilise `CROSS JOIN`

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes
CROSS JOIN vehicules;
```

MySQL

Pour le produit cartésien, on utilise `CROSS JOIN`

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes
CROSS JOIN vehicules;
```

Ou implicitement

```
SELECT nom, prenom, ville, marque, modele
FROM  personnes, vehicules;
```

MySQL

Exercice 10

Écrire une requête **SQL** qui permet d'afficher le résultat de la jointure entre personne et véhicule + les personnes n'ayant pas de voiture + les voitures n'ayant pas de véhicule (**full join**)

MySQL

Imbriquer les requêtes

Avoir une requête dans la clause d'une autre.

MySQL

Imbriquer les requêtes

Avoir une requête dans la clause d'une autre.

Plusieurs niveaux d'imbrication

- `WHERE`
- `FROM`
- `HAVING`

MySQL

Exercice 11

Écrire une requête **SQL** qui permet de sélectionner les personnes qui ont à la fois un véhicule Fiat et un Peugeot

MySQL

Pour répondre à la question de l'exercice 7, on peut utiliser les requêtes imbriquées

```
SELECT nom, prenom
FROM  personnes, vehicules
WHERE  id = id_personne
AND marque = 'Fiat'
AND id IN (SELECT id_personne
            FROM  vehicules
            WHERE   marque = 'Peugeot');
```

Le résultat

```
+------+--------+
| nom  | prenom |
+------+--------+
| Wick | John   |
+------+--------+
```

MySQL

La même requête peut être réécrite avec `= any`

```
SELECT nom, prenom
FROM  personnes, vehicules
WHERE  id = id_personne
AND marque = 'Fiat'
AND id = ANY (SELECT id_personne
              FROM  vehicules
              WHERE   marque = 'Peugeot');
```


Le résultat

```
+------+--------+
| nom  | prenom |
+------+--------+
| Wick | John   |
+------+--------+
```

MySQL

SOME est le synonyme de ANY

```
SELECT nom, prenom
FROM  personnes, vehicules
WHERE  id = id_personne
AND marque = 'Fiat'
AND id = SOME (SELECT id_personne
               FROM  vehicules
               WHERE   marque = 'Peugeot');
```


Le résultat

```
+------+--------+
| nom  | prenom |
+------+--------+
| Wick | John   |
+------+--------+
```

MySQL

Pour les requêtes imbriquées, on peut utiliser

- `IN`
- `ALL`
- `ANY` **ou** `SOME`

MySQL

Pour les requêtes imbriquées, on peut utiliser

- `IN`
- `ALL`
- `ANY` ou `SOME`

Avec `ALL` et `IN`, on peut utiliser les opérateurs de comparaison suivants

=, <, >, <>, !=, <=, >=, <, >, ! > ou ! <.

MySQL

Exercice 12

Écrire une requête **SQL** qui permet de sélectionner les marques de voiture qui appartiennent à des personnes ayant deux véhicules et dont la ville = Paris.

MySQL

La même requête peut être réécrite avec `exists` qui retourne un booléen

```
SELECT nom, prenom
FROM  personne p, vehicule v
WHERE  id = id_personne
AND marque = 'Fiat'
AND EXISTS (SELECT *
            FROM  vehicules v2
            WHERE   marque = 'Peugeot'
            AND v.id_personne = v2.id_personne);
```


Le résultat est le même

```
+------+--------+
| nom  | prenom |
+------+--------+
| Wick | John   |
+------+--------+
```

MySQL

Pour les requêtes imbriquées, il est aussi possible d'utiliser la négation

- `NOT IN`
- `NOT EXISTS`
- ...

MySQL

On peut aussi imbriquer des requêtes dans la clause `from`

```
SELECT nom, prenom
FROM  personnes p, (SELECT id_personne FROM vehicules WHERE
   marque = 'Peugeot') AS peugeot
WHERE   peugeot.id_personne = p.id;
```

Le résultat

```
+---------+--------+
| nom     | prenom |
+---------+--------+
| Benamar | Pierre |
| Wick    | John   |
+---------+--------+
```

MySQL

Exercice 13

Écrire une requête **SQL** qui permet de sélectionner le nom et le prénom de chaque personne ainsi que son nombre de véhicule.

MySQL

Exercice 13

Écrire une requête **SQL** qui permet de sélectionner le nom et le prénom de chaque personne ainsi que son nombre de véhicule.

Exercice 14

Écrire une requête **SQL** qui permet de sélectionner le nom et le prénom de chaque personne ayant au moins deux véhicules.

MySQL

Exercice 13

Écrire une requête **SQL** qui permet de sélectionner le nom et le prénom de chaque personne ainsi que son nombre de véhicule.

Exercice 14

Écrire une requête **SQL** qui permet de sélectionner le nom et le prénom de chaque personne ayant au moins deux véhicules.

Exercice 15

Écrire une requête **SQL** qui permet de sélectionner le nom et le prénom des personnes ayant le plus grand nombre de véhicules.

MySQL

Plusieurs fonctions prédéfinies

- `CONCAT`
- `TRIM`
- `LENGTH`
- `SUBSTRING`
- `UPPER`
- `LOWER`
- ...

MySQL

Exemple avec `concat` et `upper`

```
SELECT CONCAT(UPPER(nom), prenom) AS nom_complet
FROM  personnes;
```

Le résultat

```
+----------------+
| nom_complet    |
+----------------+
| COHENSophie    |
| LEBERREBernard |
| BENAMARPierre  |
| HADADKarim     |
| WICKJohn       |
+----------------+
```

MySQL

Plusieurs fonctions prédéfinies

- `ABS`
- `CEIL`
- `FLOOR`
- `POWER`
- `ROUND`
- `SQRT`
- ...

MySQL

Trois méthodes pour récupérer la date actuelle (sans l'heure)

```
SELECT CURDATE(), CURRENT_DATE(), CURRENT_DATE;
```

MySQL

Trois méthodes pour récupérer la date actuelle (sans l'heure)

```
SELECT CURDATE(), CURRENT_DATE(), CURRENT_DATE;
```

Trois méthodes pour récupérer l'heure actuelle (sans la date)

```
SELECT CURTIME(), CURRENT_TIME(), CURRENT_TIME;
```

MySQL

Deux méthodes pour récupérer la date et heure actuelles

```
SELECT NOW(), SYSDATE();
```

MySQL

Deux méthodes pour récupérer la date et heure actuelles

```
SELECT NOW(), SYSDATE();
```

Pour récupérer le Timestamp Unix

```
SELECT UNIX_TIMESTAMP();
```

MySQL

Il est possible de stocker le résultat d'une requête dans une variable en utilisant la clause `INTO`

```
SELECT salaire INTO @salaire FROM personnes WHERE id=1;
```

MySQL

Il est possible de stocker le résultat d'une requête dans une variable en utilisant la clause `INTO`

```
SELECT salaire INTO @salaire FROM personnes WHERE id=1;
```

Pour afficher le contenu d'une `@variable`

```
SELECT @salaire;
```

MySQL

Une `@variable` peut être utilisée dans une autre requête

```
SELECT *
FROM personnes
WHERE salaire=@salaire;
```

MySQL

Une `@variable` peut être utilisée dans une autre requête

```
SELECT *
FROM personnes
WHERE salaire=@salaire;
```

Résultat

```
+-----+-------+--------+---------+-----------+
| id  | nom   | prenom | salaire | ville     |
+-----+-------+--------+---------+-----------+
|   1 | Cohen | Sophie |    2000 | Marseille |
+-----+-------+--------+---------+-----------+
```

MySQL

Les `@variable` en **MySQL** ont certaines limites à prendre en compte

- Le nom d'une `@variable` commence par un signe @, suivi d'un identificateur.
- Le nom de la variable doit être unique dans le cadre de la session **MySQL** en cours.
- Les `@variable` ne sont accessibles que pendant la session **MySQL** en cours.
- Deux sessions différentes ne partagent pas les mêmes `@variable`.
- Les `@variable` sont généralement utilisées pour stocker des valeurs uniques, telles que des entiers, des chaînes de caractères... et non des ensembles de résultats (colonne).
- ...

MySQL

On peut aussi initialiser une `@variable` directement avec `SET`

```
SET @salaire = 3000;
```

MySQL

On peut aussi initialiser une `@variable` directement avec `SET`

```
SET @salaire = 3000;
```

Ou avec `SELECT` (`SELECT` avec l'opérateur = effectue une comparaison)

```
SELECT @salaire := 3000;
```

MySQL

Solution

Et si on avait un modèle requête qu'on exécute souvent : comme par exemple, chercher une personne selon le salaire.

MySQL

Solution

Et si on avait un modèle requête qu'on exécute souvent : comme par exemple, chercher une personne selon le salaire.

Solution

- Définir le modèle de la requête avec `PREPARE`
- Exécuter la requête avec `EXECUTE`

MySQL

Exemple de préparation de requête

```
PREPARE personne_par_salaire
FROM 'SELECT * FROM personnes WHERE salaire = ?';
```

MySQL

Exemple de préparation de requête

```
PREPARE personne_par_salaire
FROM 'SELECT * FROM personnes WHERE salaire = ?';
```

Exemple d'exécution

```
SET @salaire = 3000;
EXECUTE personne_par_salaire USING @salaire;
```

MySQL

Exemple de préparation de requête

```
PREPARE personne_par_salaire
FROM 'SELECT * FROM personnes WHERE salaire = ?';
```

Exemple d'exécution

```
SET @salaire = 3000;
EXECUTE personne_par_salaire USING @salaire;
```

Suppression d'une requête préparée

```
DEALLOCATE PREPARE personne_par_salaire;
```

MySQL

Requête préparée : inconvénient

Stockée en mémoire pour la session courante.

MySQL

Requête préparée : inconvénient

Stockée en mémoire pour la session courante.

Requête préparée : avantages

- Se protéger contre les injections **SQL**.
- Améliorer la performance si la requête est exécutée plusieurs fois par la même session.

MySQL

Fenêtres (window functions)

- Permettent d'effectuer des calculs sur un ensemble de lignes
- Sans regrouper les résultats comme avec `GROUP BY`.
- Souvent utilisées pour fournir des statistiques comme
  - Calculer des cumuls (**cumulative sum**) sans regrouper les lignes.
  - Attribuer un rang ou une position dans un ensemble de données.
  - Calculer des moyennes glissantes (**moving average**) ou des statistiques par groupes.

MySQL

Syntaxe général

```
fonction_fenêtre() OVER ([PARTITION BY colonne] [ORDER BY colonne])
```

- `fonction_fenêtre` : `SUM()`, `AVG()`, `ROW_NUMBER()`, `RANK()`, `DENSE_RANK()`...
- `PARTITION BY` : divise les données en sous-ensembles.
- `ORDER BY` : définit l'ordre d'application de la fonction.

MySQL

Pour classer les personnes en fonction de leur salaire

```
SELECT *, RANK() OVER(ORDER BY salaire DESC) AS rang_salaire
FROM personnes;
```

Le résultat

```
+-----+---------+---------+---------+-----------+--------------+
| id  | nom     | prenom  | salaire | ville     | rang_salaire |
+-----+---------+---------+---------+-----------+--------------+
|   5 | Wick    | John    |    3000 | Paris     |            1 |
|   4 | Hadad   | Karim   |    2500 | Paris     |            2 |
|   1 | Cohen   | Sophie  |    2000 | Marseille |            3 |
|   3 | Benamar | Pierre  |    1800 | Lyon      |            4 |
|   2 | Leberre | Bernard |    1500 | Marseille |            5 |
+-----+---------+---------+---------+-----------+--------------+
```

MySQL

Pour classer les personnes en fonction de leur salaire, ville par ville

```
SELECT *,  RANK() OVER(PARTITION BY ville ORDER BY salaire DESC) AS
   rang_salaire
FROM personnes;
```

Le résultat

```
+-----+---------+---------+---------+-----------+--------------+
| id  | nom     | prenom  | salaire | ville     | rang_salaire |
+-----+---------+---------+---------+-----------+--------------+
|   3 | Benamar | Pierre  |    1800 | Lyon      |            1 |
|   1 | Cohen   | Sophie  |    2000 | Marseille |            1 |
|   2 | Leberre | Bernard |    1500 | Marseille |            2 |
|   5 | Wick    | John    |    3000 | Paris     |            1 |
|   4 | Hadad   | Karim   |    2500 | Paris     |            2 |
+-----+---------+---------+---------+-----------+--------------+
```

MySQL

Pour compter les personnes par ville

```
SELECT *, COUNT(*) OVER(PARTITION BY ville) AS nb_pers_par_ville
FROM personnes;
```

Le résultat

```
+-----+---------+---------+---------+-----------+--------------------+
| id  | nom     | prenom  | salaire | ville     | nb_pers_par_ville  |
+-----+---------+---------+---------+-----------+--------------------+
|   3 | Benamar | Pierre  |    1800 | Lyon      |                  1 |
|   1 | Cohen   | Sophie  |    2000 | Marseille |                  2 |
|   2 | Leberre | Bernard |    1500 | Marseille |                  2 |
|   4 | Hadad   | Karim   |    2500 | Paris     |                  2 |
|   5 | Wick    | John    |    3000 | Paris     |                  2 |
+-----+---------+---------+---------+-----------+--------------------+
```

MySQL

Pour classer les personnes en fonction de leur salaire

```
SELECT *, ROW_NUMBER() OVER(ORDER BY salaire DESC) AS rang_salaire
FROM personnes;
```

Le résultat

```
+-----+---------+---------+---------+-----------+--------------+
| id  | nom     | prenom  | salaire | ville     | rang_salaire |
+-----+---------+---------+---------+-----------+--------------+
|   5 | Wick    | John    |    3000 | Paris     |            1 |
|   4 | Hadad   | Karim   |    2500 | Paris     |            2 |
|   1 | Cohen   | Sophie  |    2000 | Marseille |            3 |
|   3 | Benamar | Pierre  |    1800 | Lyon      |            4 |
|   2 | Leberre | Bernard |    1500 | Marseille |            5 |
+-----+---------+---------+---------+-----------+--------------+
```

MySQL

Pour classer les personnes en fonction de leur salaire

```
SELECT *, ROW_NUMBER() OVER(ORDER BY salaire DESC) AS rang_salaire
FROM personnes;
```

Le résultat

```
+-----+---------+---------+---------+-----------+--------------+
| id  | nom     | prenom  | salaire | ville     | rang_salaire |
+-----+---------+---------+---------+-----------+--------------+
|   5 | Wick    | John    |    3000 | Paris     |            1 |
|   4 | Hadad   | Karim   |    2500 | Paris     |            2 |
|   1 | Cohen   | Sophie  |    2000 | Marseille |            3 |
|   3 | Benamar | Pierre  |    1800 | Lyon      |            4 |
|   2 | Leberre | Bernard |    1500 | Marseille |            5 |
+-----+---------+---------+---------+-----------+--------------+
```

ROW_NUMBER et RANK font-elles la même chose ?

MySQL

ROW_NUMBER

- Numérotation séquentielle sans se soucier des égalités.
- Si deux lignes ont la même valeur de tri, elles recevront des numéros différents.
- Le classement est toujours séquentiel : pas de `trou` dans la numérotation.

RANK

- Classement avec des égalités dans la numérotation.
- Si deux lignes ont la même valeur de tri, elles recevront les mêmes numéros.
- Le rang suivant saute les positions en fonction du nombre de lignes identiques : on parle de `trous` dans la séquence.

MySQL

ROW_NUMBER

- Numérotation séquentielle sans se soucier des égalités.
- Si deux lignes ont la même valeur de tri, elles recevront des numéros différents.
- Le classement est toujours séquentiel : pas de `trou` dans la numérotation.

RANK

- Classement avec des égalités dans la numérotation.
- Si deux lignes ont la même valeur de tri, elles recevront les mêmes numéros.
- Le rang suivant saute les positions en fonction du nombre de lignes identiques : on parle de `trous` dans la séquence.

Et DENSE_RANK ?

MySQL

DENSE_RANK

- Similaire à `RANK`, mais sans **trous** dans la séquence.
- Attribue le même rang aux lignes ayant la même valeur, **mais** le rang suivant est incrémenté de 1 sans sauter de positions.

MySQL

DENSE_RANK

- Similaire à `RANK`, mais sans **trous** dans la séquence.
- Attribue le même rang aux lignes ayant la même valeur, **mais** le rang suivant est incrémenté de 1 sans sauter de positions.

Pour récapituler

- `ROW_NUMBER` : numérotation séquentielle (`1, 2, 3, 4`).
- `RANK` : les ex-æquo partagent le même rang, ensuite saut de positions (`1, 1, 3, 4`).
- `DENSE_RANK` : les ex-æquo partagent le même rang, sans saut après (`1, 1, 2, 3`).

MySQL

Il est possible d'utiliser plusieurs fonctions fenêtres dans une même requête

```
SELECT
    *,
    ROW_NUMBER() OVER(PARTITION BY ville ORDER BY salaire DESC) AS rang,
    SUM(salaire) OVER(PARTITION BY ville) AS total_ville
FROM personnes;
```

Le résultat

```
+-----+---------+---------+---------+-----------+------+--------------+
| id  | nom     | prenom  | salaire | ville     | rang | total_ville  |
+-----+---------+---------+---------+-----------+------+--------------+
|   3 | Benamar | Pierre  |    1800 | Lyon      |    1 |         1800 |
|   1 | Cohen   | Sophie  |    2000 | Marseille |    1 |         3500 |
|   2 | Leberre | Bernard |    1500 | Marseille |    2 |         3500 |
|   5 | Wick    | John    |    3000 | Paris     |    1 |         5500 |
|   4 | Hadad   | Karim   |    2500 | Paris     |    2 |         5500 |
+-----+---------+---------+---------+-----------+------+--------------+
```

MySQL

Pour éviter de répéter deux fois `PARTITION BY ville`, on utilise la clause `WINDOW`

```
SELECT
    *,
    ROW_NUMBER() OVER w AS rang,
    SUM(salaire) OVER w AS total_ville
FROM personne
WINDOW w AS (PARTITION BY ville ORDER BY salaire DESC);
```

Le résultat

id	nom	prenom	salaire	ville	rang	total_ville
3	Benamar	Pierre	1800	Lyon	1	1800
1	Cohen	Sophie	2000	Marseille	1	2000
2	Leberre	Bernard	1500	Marseille	2	3500
5	Wick	John	3000	Paris	1	3000
4	Hadad	Karim	2500	Paris	2	5500