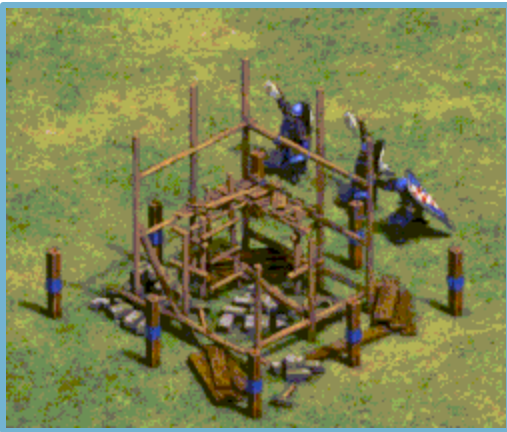


Rencontre 5

Requêtes

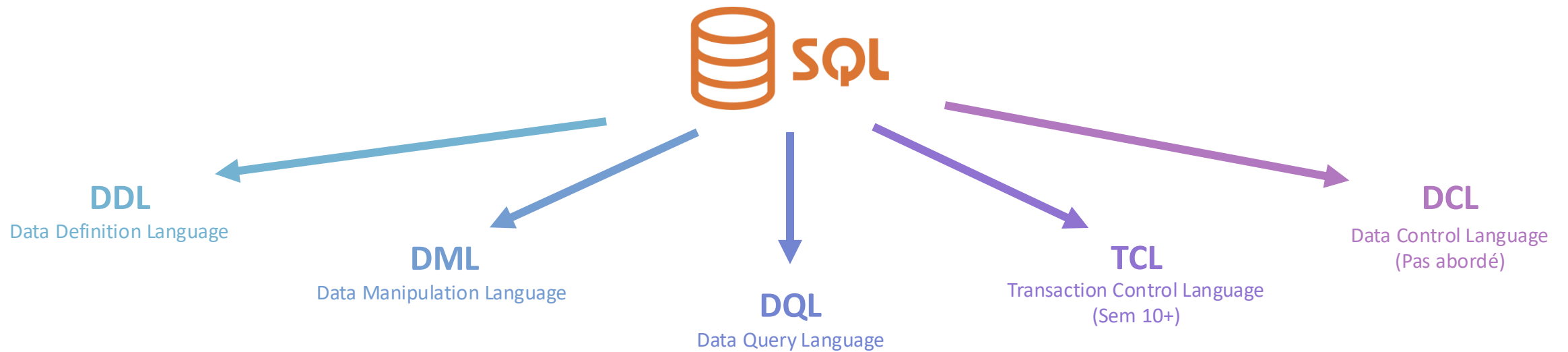
Bases de données et programmation Web



On va finalement *construire* la base de données !



❖ Requête de données (DQL)





❖ Requête de données

◆ La famille **DQL** (Data Query language) ne comporte qu'une seule instruction : **SELECT** ! Mais ... comme vous le savez déjà, SELECT inclut des tonnes de possibilités :

- Sélection simple
- WHERE (Conditions)
- ORDER BY (Ordonnancement)
- Agrégation
- Renommage
- Jointures
- EXISTS
- Any
- All
- Sous-requêtes
- Fonctions et opérations sur colonnes



❖ Sélection simple

```
SELECT Colonne1, Colonne2, ... FROM Table1;
```

```
SELECT * FROM Table1;
```

Pour sélectionner **toutes** les colonnes facilement.

```
SELECT DISTINCT Colonne1, Colonne2, ... FROM Table1;
```

Pour sélectionner seulement les **ensembles différents**. (Ex : pratique si on prend peu de colonnes sans la clé primaire)

```
SELECT PersonnageID, Nom FROM Personnages.Personnage;
```



PersonnageID	Nom
9	Bébé Mario
4	Bowser
12	Bowser Jr.
8	Koopa
2	Luigi
5	Mario
3	Peach
6	Wario
7	Yoshi



❖ Where (conditions)

```
SELECT colonne1, colonne2, ... FROM Table1  
WHERE condition;
```

◆ Opérateurs de comparaison :

colonne1 > 5

colonne2 = 'Voiture'

colonne1 != 2

colonne1 <= 10.5

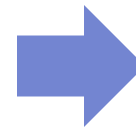
colonne1 **BETWEEN** 1 AND 5

colonne1 **IN** ('a', 'b', 'c')

colonne1 doit valoir 1, 2, 3, 4 ou 5

colonne1 doit valoir a, b ou c

```
SELECT * FROM Personnages.Personnage  
WHERE Poids = 'lourd'
```



PersonnageID	Nom	Poids	Deverrouille
4	Bowser	lourd	1
6	Wario	lourd	1



❖ Where (conditions)

```
SELECT colonne1, colonne2, ... FROM Table1  
WHERE condition;
```

◆ **LIKE**: Opérateur de comparaison sur les chaînes de caractères :

% veut dire 0 à N caractères au choix.

_ veut dire exactement 1 caractère au choix.

colonne3 **LIKE** 'a%'

colonne4 **LIKE** '%b'

colonne5 **LIKE** '%c%'

colonne6 **LIKE** '_%d'

colonne7 **LIKE** '%e__'

Exemples avec **LIKE** :

- **colonne3** doit être une chaîne de caractères qui commence par **a**.
- **colonne4** doit être une chaîne de caractères qui se termine par **b**.
- **colonne5** doit être une chaîne de caractères qui contient au moins un **c**, n'importe où.
- **colonne6** doit se terminer par **d**. Par contre, le **d** doit être précédé d'au moins un caractère quelconque.
- **colonne7** doit se terminer par **e** suivi d'exactement deux caractères quelconques.



❖ Where (conditions)

◆ Opérateurs logiques

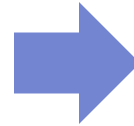
```
condition1 OR condition2 AND condition3
```

Attention : L'opérateur **AND** est prioritaire sur l'opérateur **OR**. (Si on n'utilise pas de parenthèses pour changer la priorité)

```
NOT condition1
```

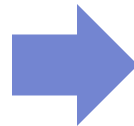
NOT inverse le résultat de la condition. (Comme **!** en C#)

```
SELECT * FROM Personnages.Personnage  
WHERE Poids = 'moyen' OR EstDeverrouille=0;
```



PersonnageID	Nom	Poids	EstDeverrouille
1	Mario	moyen	1
2	Luigi	moyen	1
3	Peach	moyen	1
4	Yoshi	moyen	1
5	Bowser Jr.	moyen	0
6	Daisy	moyen	0
7	Toad	léger	0

```
SELECT * FROM Personnages.Personnage  
WHERE Poids != 'moyen'
```



PersonnageID	Nom	Poids	EstDeverrouille
7	Toad	léger	0
8	Koopa	léger	1
9	Bébé Mario	léger	0
10	Bébé Luigi	léger	0



❖ ORDER BY

```
SELECT Colonne1, Colonne2, ... FROM Table1  
ORDER BY Colonne1;
```

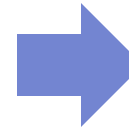
Tri **croissant** par rapport à la colonne1.

```
SELECT Colonne1, Colonne2, ... FROM Table1  
WHERE Colonne1 < 5  
ORDER BY Colonne1 DESC, Colonne2 DESC;
```

Tri **décroissant** par rapport à la colonne1, et si des valeurs sont identiques dans la colonne1, alors par rapport à la colonne2.

```
SELECT Nom, Vitesse, Acceleration  
FROM Courses.Kart  
ORDER BY Vitesse ASC, Acceleration ASC;
```

La liste des Karts, classés par vitesse croissante. Si des Karts ont la même vitesse, on les classe par accélération croissante.



Nom	Vitesse	Acceleration
Scootinette	1.00	5.00
Quad Wiggler	1.50	4.50
Paracoccinelly	2.25	4.00
Caravéloce	2.50	3.75
Rétro	2.75	3.50
Beat-bolide	3.00	3.00
Autorhino	3.00	3.25
Chabriolet	3.25	3.00



❖ ORDER BY

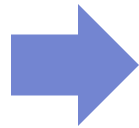
```
SELECT TOP (5) colonne1, colonne2, ...  
FROM Table1  
ORDER BY Colonne1 DESC;
```

Sélectionne seulement les 5 premières rangées obtenues. **TOP** est utilisable principalement avec **ORDER BY**, pour faire une sélection intéressante. Dans ce cas-ci, on sélectionne les 5 rangées avec les plus grandes valeurs pour colonne1.

```
SELECT TOP (1) * FROM Table1  
ORDER BY Table1ID DESC;
```

Sélectionne le dernier enregistrement de la table Table1

```
SELECT TOP 3 Nom, Vitesse  
FROM Courses.Kart  
ORDER BY Vitesse DESC;
```



Nom	Vitesse
Blue Falcon	5.00
Koopa-mobile	3.25
Chabriolet	3.25

Les trois Karts avec la meilleure vitesse.



❖ Agrégation

```
SELECT MIN(Colonne1) FROM Table1
```

```
SELECT MAX(Colonne2) FROM Table1  
WHERE condition;
```

Obtenir le minimum ou le maximum pour les valeurs d'une colonne. (Avec ou sans condition) Il y a également les fonctions **AVG()**, **COUNT()** et **SUM()** pour obtenir respectivement la moyenne des valeurs, le nombre de valeurs et la somme des valeurs.

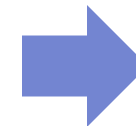
```
SELECT MAX(Vitesse)  
FROM Courses.Kart
```



(Aucun nom de colonne)
5.00

La valeur maximale de Vitesse pour toutes les données de la table.

```
SELECT MIN(TenueDeRoute) AS [Min Tenue de Route]  
FROM Courses.Kart
```



Min Tenue de Route
1.50

La valeur minimale de TenueDeRoute pour toutes les données de la table.



❖ Agrégation

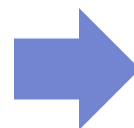
```
SELECT Colonne2 , MIN(Colonne1)
FROM Table1
GROUP BY Colonne2
```

Pour chaque valeur de Colonne 2, obtenir le minimum de la Colonne1 parmi toutes les rangées ayant cette valeur de colonne2. (Ex : parmi toutes les rangées qui ont la valeur 'Pneu' dans colonne2, le minimum de colonne1 est 50)
Fonctionnement identique avec MAX(), AVG(), COUNT() et SUM().

```
SELECT Colonne2, Colonne3 , SUM(Colonne1)
FROM Table1
WHERE Colonne3 > 100
GROUP BY Colonne2, Colonne3
```

Quand on utilise une fonction d'agrégation (comme SUM), toutes les autres colonnes (ici, colonne2 et colonne3) sélectionnées doivent absolument se retrouver dans le GROUP BY, ou dans une autre fonction d'agrégation, sinon ça ne fonctionnera pas.

```
SELECT Poids, COUNT(PersonnageID) AS [NbPersonnages]
FROM Personnages.Personnage
GROUP BY Poids
```



Poids	NbPersonnages
léger	5
lourd	4
moyen	9

Pour chaque poids (léger, moyen et lourd), on compte le nombre de personnages.

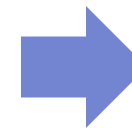


❖ Agrégation

```
SELECT Colonne2 , COUNT(Colonne1) AS [Alias De Champ] FROM Table1  
WHERE condition  
GROUP BY Colonne2  
HAVING COUNT(Colonne1) > 5
```

Avec **WHERE**, on sélectionne des rangées avant l'agrégation, mais on ne peut pas sélectionner des données obtenues grâce à une fonction d'agrégation. Avec **HAVING**, on peut sélectionner des rangées qui satisfont aux valeurs obtenues par les fonctions d'agrégation.

```
SELECT Poids, COUNT(PersonnageID) AS [NbPersonnages]  
FROM Personnages.Personnage  
GROUP BY Poids  
HAVING COUNT(PersonnageID) <= 5
```



Poids	NbPersonnages
léger	5
lourd	4

Pour chaque poids (léger, moyen et lourd), on compte le nombre de rangées, (Donc le nombre de personnage) mais on affiche seulement les données où le nombre de rangées compté est inférieur à 5.



❖ **Standard** de renommage avec un ALIAS: **AS** [le nouveau nom]

```
SELECT Colonne1 AS [Nom1], Colonne2, Colonne3 AS [Nom2] FROM Table1;
```

Dans le résultat de la requête, colonne1 et colonne3 auront des noms différents. Parfois nécessaire pour éviter les conflits lors d'une jointure. Si les nouveaux noms incluent des espaces, on peut les encadrer avec des crochets ou des apostrophes.

```
SELECT Poids, COUNT(PersonnageID) AS [Nombre de personnages]  
FROM Personnages.Personnage  
GROUP BY Poids
```



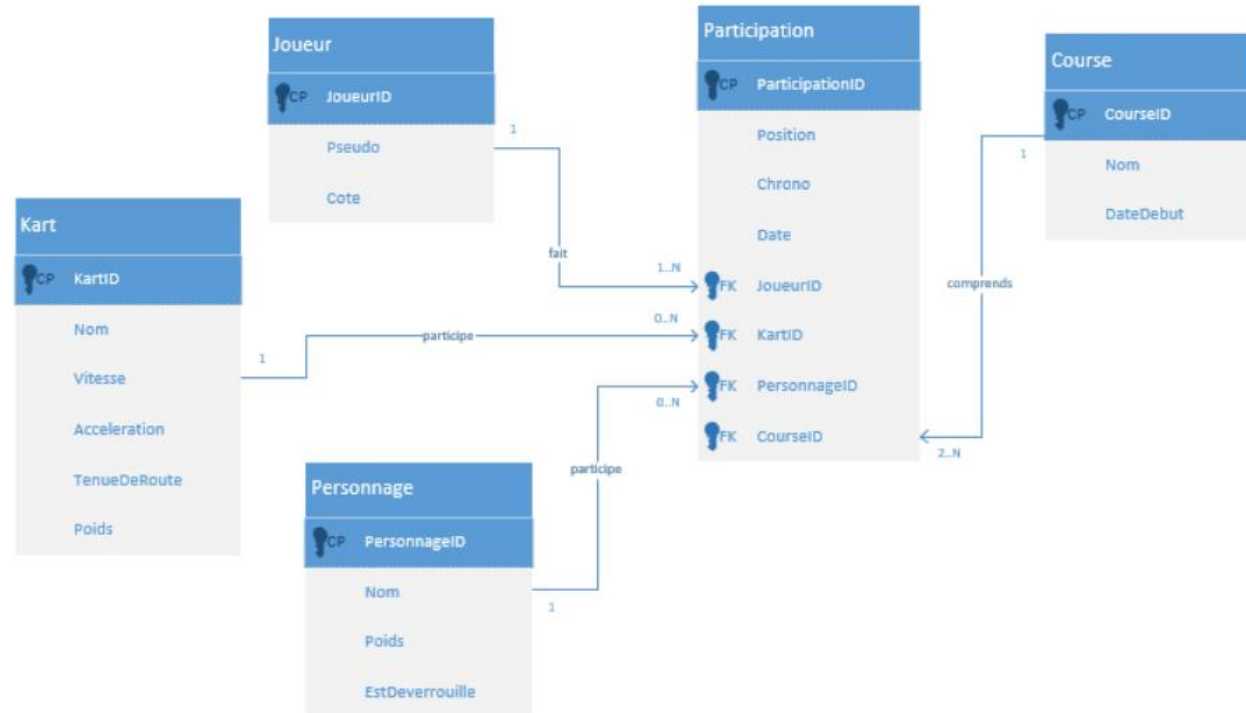
Poids	Nombre de personnages
léger	5
lourd	4
moyen	9



❖ Jointures

- ◆ Les jointures permettent de faire des requêtes sur plusieurs tables. Les exemples des prochaines diapositives utiliseront les tables de ce schéma :

À chaque fois qu'un **joueur participe** à une **cOURSE** dans un jeu multijoueur de Karting, il choisit un **personnage** et un **kart**. Il obtient alors une **position** et un **chrono** qui lui sont associés dans la table **Participation**.





❖ Jointures , AVEC ALIAS de TABLES

```
SELECT Colonne1, Colonne2, ...  
FROM Joueur J  
INNER JOIN Partie P  
ON J.JoueurID = P.JoueurID
```

Jointure des deux tables basées sur une paire de colonnes.

On renomme **TOUJOURS** les tables lors des jointures pour simplifier certaines parties de la requête ou préciser de quelle table est tirée chaque colonne sélectionnée.

On omet habituellement le 'as' entre la table et son alias.

On utilise la première lettre ou les premières lettres (au besoin) de nom des tables. Pas de T1,T2, X, Y, Z...

Si on veut faire une jointure entre les tables Client et Commande, on utiliserait les alias CL et CO.



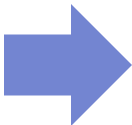
❖ Jointures , AVEC ALIAS de TABLES

```
SELECT Pseudo, Position
FROM Joueurs.Joueur J
INNER JOIN Courses.Participation P
ON J.JoueurID = P.JoueurID
WHERE Pseudo = [MonkeyDriver]
```

Ici, on a sélectionné toutes les participations de MonkeyDriver pour afficher son pseudo et la position qu'il a obtenue à chaque participation. Nous avons utilisé **JoueurID** pour établir la correspondance entre le **Joueur** et ses **Participations**.

JoueurID	Pseudo	Cote
1	ProgamerZ1	2997
2	N00bie	512
3	LuigiTime	1489
4★	MonkeyDriver	2501
5	Drunkart	789
6	SammySam	1322
7	Benna4	1663
8	boifem_t	2667

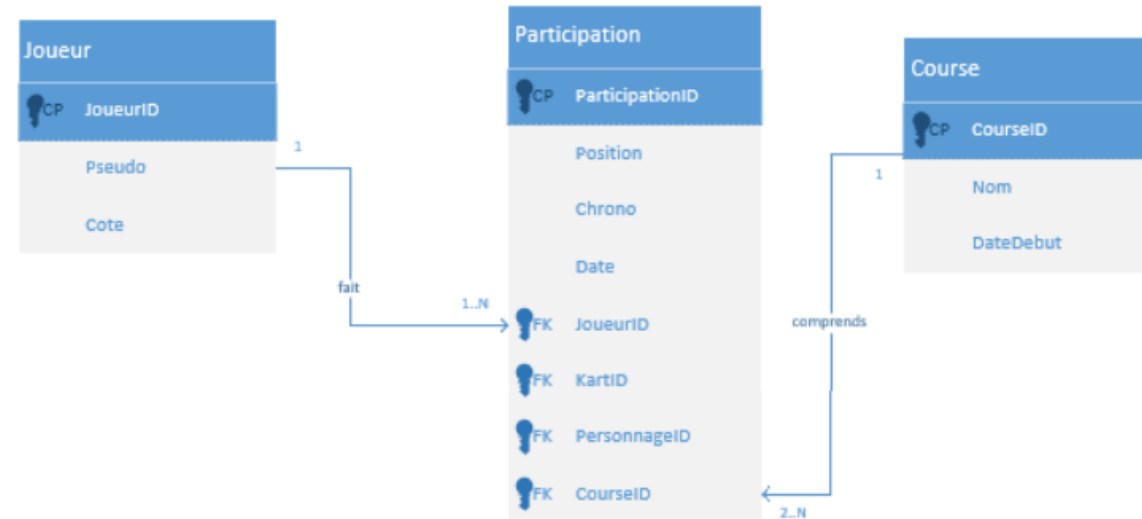
ParticipationID	Position	Chrono	JoueurID	PersonnageID	KartID	CourselD	date
2	12	9371	2	5	2	9	2023-01-31 17:21:06.347
3	2	7820	1	2	7	2	2023-01-31 17:21:06.347
4	4	8742	7	8	4	8	2023-01-31 17:21:06.347
5	5	3719	4★	3	1	4	2023-01-31 17:21:06.347
6	1	4829	1	9	3	8	2023-01-31 17:21:06.347
8	1	6378	1	7	10	3	2023-01-31 17:21:06.347
9	1	8791	1	3	7	10	2023-01-31 17:21:06.347
10	12	4251	2	9	7	3	2023-01-31 17:21:06.347
11	6	4322	4★	3	1	5	2023-01-31 17:21:06.347
12	9	8762	6	4	6	2	2023-01-31 17:21:06.347
13	1	8381	7	8	3	4	2023-01-31 17:21:06.347
19	12	8532	5	5	9	7	2023-01-31 17:21:06.347



Pseudo	Position
MonkeyDriver	5
MonkeyDriver	6



❖ Jointures, AVEC 3 tables



```
SELECT J.Pseudo, P.Position, C.Nom
FROM Joueurs.Joueur J
INNER JOIN Courses.Participation P
ON J.JoueurID = P.JoueurID
INNER JOIN Courses.Course C
ON C.CourseID = P.CourseID
```

Pseudo	Position	Nom
LittleBroStoleMyRemote	6	Koopapolis
veryN00b	6	Horloge Tic-Tac
PleaseDontTheCat	2	Mine Wario
NeverGonnaGiveYouUp	4	Aéroport Azur
AndrewTaaaaa	6	Désert Toussec
proracerdude29	4	Mine Wario
Rickroll69	7	Piste musicale
RageQuitGuy	6	Piste musicale

Cette fois, on a fait une jointure sur trois tables : **Joueurs**, **Participations** et **Courses**. En résumé, pour toutes les participations, on est allé récupérer le pseudo associé au **JoueurID** dans la table Joueurs et le nom de la course associée à **CourseID** dans la table Course.



❖ Jointures

```
SELECT Colonne1, Colonne2, ...  
FROM Table1 T1  
INNER JOIN Table2 T2  
ON T1.Colonne3 = T2.Colonne3 AND T1.Colonne4 = T2.Colonne4
```

Jointure des deux tables basées sur deux paires de colonnes.

On n'a pas d'exemple intéressant dans notre BD Mario_Kart à présenter ici.

C'est utilisé principalement quand on a une jointure sur une table associative qui a une clé primaire composée de deux champs. Ce qui est de moins en moins courant de nos jours.

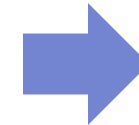
Note: ici on n'a pas d'exemples avec des vrais noms de tables alors on a utilisé T1 et T2 comme alias mais ce n'est pas une bonne pratique...



❖ Jointures **LEFT JOIN**

Avec **LEFT JOIN** la jointure sélectionne même les rangées qui n'ont pas de correspondances dans l'autre table.

```
SELECT Nom, Position, Chrono
FROM Courses.Kart K
LEFT JOIN Courses.Participation P
ON K.KartID = P.KartID
```



Nom	Position	Chrono
Kart Standard	5	3719
Kart Standard	6	4322
Chabriolet	12	9371
Kart Rétro	1	4829
Kart Rétro	1	8381
Koopa-mobile	4	8742
Blue Falcon	NULL	NULL
Autorhino	9	8762
Scootinette	2	7820
Scootinette	1	8791
Scootinette	12	4251
Yoshimoto	NULL	NULL
Quad Wiggler	12	8532
Quad Standard	1	6378

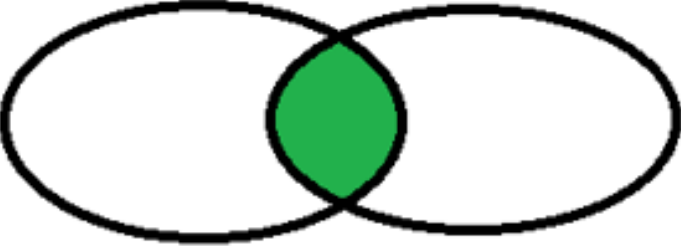
Pour tous les karts, on a récupéré les **positions** et les **chronos** qui leur sont associés dans la table **Participation**.

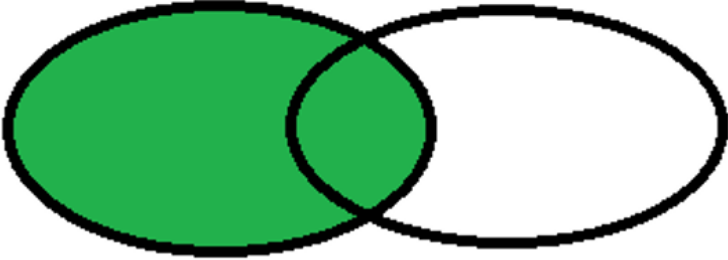
On remarque que pour deux karts, Blue Falcon et Yoshimoto, les champs position et chrono sont **NULL**. Ça veut dire qu'aucune participation n'a pu être trouvée pour ces deux karts.

De plus, si un kart apparaît plus d'une fois, ça veut dire que plusieurs participations lui sont associées.

Comparaison INNER JOIN et LEFT JOIN



Client	Facture
	
FROM Clients.Client C INNER JOIN Commandes.Facture F ON C.ClientID = F.ClientID	

Client	Facture
	
FROM Clients.Client C LEFT JOIN Commandes.Facture F ON C.ClientID = F.ClientID	



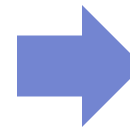
❖ Jointures

Les requêtes LEFT JOIN sont habituellement utilisées pour savoir quelles données d'une première table n'ont pas de correspondances dans la deuxième table.

On ajoute donc la clause WHERE où la clé étrangère dans la deuxième table IS NULL

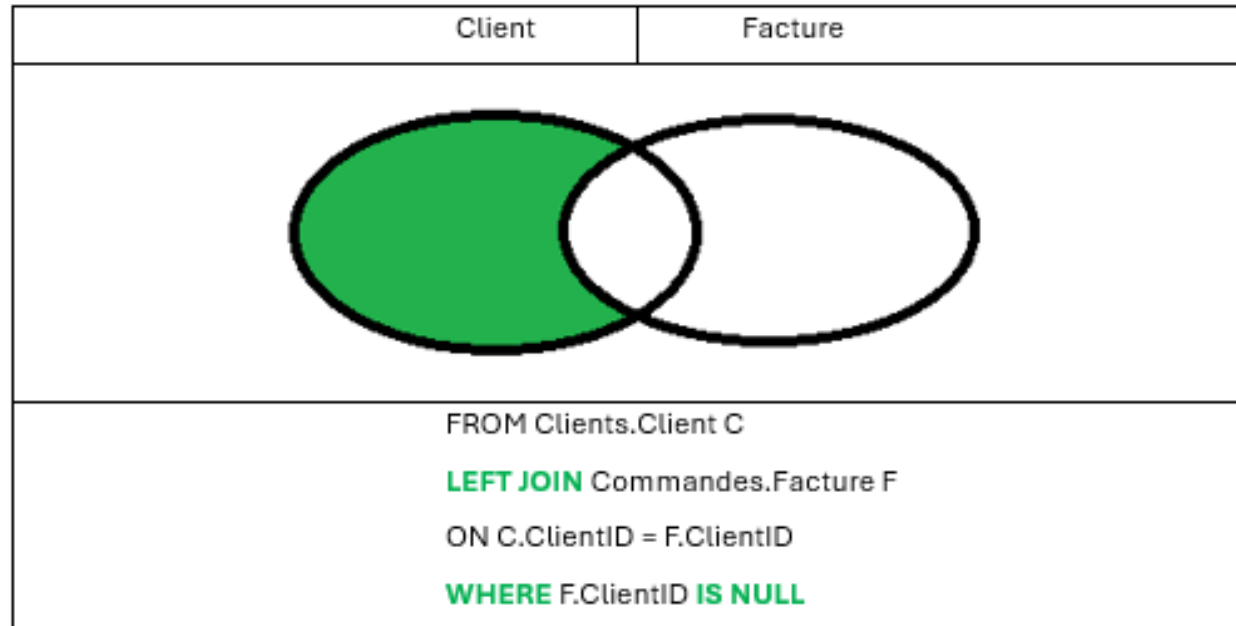
Quels karts n'ont jamais eu de participations dans la course?

```
SELECT Nom, Position, Chrono
FROM Courses.Kart K
LEFT JOIN Courses.Participation P
ON K.KartID = P.KartID
WHERE P.KartID IS NULL
```



Nom	Position	Chrono	
Yoshimoto	NULL	NULL	
Blue Falcon	NULL	NULL	

LEFT JOIN ...avec WHERE....IS NULL



Attention: Quand on utilise LEFT JOIN l'ordre des tables est très important.

Vous nommez en premier la table dont vous voulez voir les données.

Ici, voir les **clients** qui n'ont jamais eu de factures, donc **FROM Clients.Client C**



❖ Jointures

◆ Faisons une jointure sur les 5 tables

```
SELECT Pseudo, K.Nom AS [NomKart], C.Nom AS [NomCourse], Pe.Nom AS [NomPersonnage], Pa.Position, CONCAT( Pa.Chrono / 100, 's') AS [Chrono],  
FROM Courses.Participation Pa  
INNER JOIN Courses.Kart K  
ON Pa.KartID = K.KartID  
INNER JOIN Personnages.Personnage Pe  
ON Pe.PersonnageID = Pa.PersonnageID  
INNER JOIN Joueurs.Joueur J  
ON J.JoueurID = Pa.JoueurID  
INNER JOIN Courses.Course C  
ON C.CourseID = Pa.CourseID
```

Pour toutes les participations, nous sommes allés récupérer :

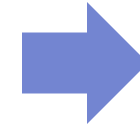
- Le pseudo du joueur dans la table Joueur.
- Le nom de la course dans la table Course.
- Le nom du personnage joué dans la table Personnage.
- Le nom du kart utilisé dans la table Kart.

Pseudo	NomKart	NomCourse	NomPersonnage	Position	Chrono
N00bie	Chabriolet	Château de Bowser	Mario	12	93s.
ProgamerZ1	Scootinette	Autodrome Royal	Luigi	2	78s.
Benna4	Koopa-mobile	Piste musicale	Koopa	4	87s.
MonkeyDriver	Kart Standard	Aéorport Azur	Peach	5	37s.
ProgamerZ1	Kart Rétro	Piste musicale	Bébé Mario	1	48s.
ProgamerZ1	Quad Standard	Autoroute Toad	Yoshi	1	63s.
ProgamerZ1	Scootinette	Route Ruban	Peach	1	87s.
N00bie	Scootinette	Autoroute Toad	Bébé Mario	12	42s.
MonkeyDriver	Kart Standard	Big Blue	Peach	6	43s.
SammySam	Autorhino	Autodrome Royal	Bowser	9	87s.
Benna4	Kart Rétro	Aéorport Azur	Koopa	1	83s.
Drunkart	Quad Wiggler	Mine Wario	Mario	12	85s.



❖ EXISTS

```
SELECT Nom
FROM Courses.Kart K
WHERE EXISTS ( SELECT ParticipationID
                FROM Courses.Participation P
                WHERE P.KartID = K.KartID)
```



Nom
Kart Standard
Chabriolet
Kart Rétro
Koopa-mobile
Autorhino
Scootinette
Quad Wiggler
Quad Standard

L'opérateur **EXISTS** permet de sélectionner seulement les rangées pour lesquelles au moins une donnée **existe dans une sous-requête**.

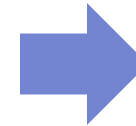
On sélectionne tous les noms de karts pour lesquels il existe au moins une participation associée au KartID.



❖ EXISTS

On peut utiliser **NOT EXISTS** pour sélectionner seulement les rangées pour lesquelles AUCUNE donnée existe dans une sous-requête.

```
SELECT Nom
FROM Courses.Kart K
WHERE NOT EXISTS ( SELECT ParticipationID
                   FROM Courses.Participation P
                   WHERE P.KartID = K.KartID)
```



Nom
Blue Falcon
Yoshimoto

On sélectionne tous les noms de karts pour lesquels il n'existe aucune participation associée au KartID.

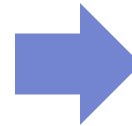


❖ Trouver tous X qui n'a pas de Y

Quels karts n'ont jamais eu de participations dans la course?

On a donc vu 2 façons de répondre à cette question:

```
SELECT Nom AS [Nom du Kart]
FROM Courses.Kart K
LEFT JOIN Courses.Participation P
ON K.KartID = P.KartID
WHERE P.KartID IS NULL
```



Nom du Kart
Blue Falcon
Yoshimoto

```
SELECT Nom AS [Nom du Kart]
FROM Courses.Kart K
WHERE NOT EXISTS ( SELECT ParticipationID
                   FROM Courses.Participation P
                   WHERE P.KartID = K.KartID)
```



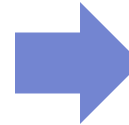
Nom du Kart
Blue Falcon
Yoshimoto



❖ Trouver tous X qui n'a pas de Y

L'utilisation du LEFT JOIN avec la clause WHERE où la clé de la deuxième table est NULL est la méthode à privilégier pour répondre à ce type de question car c'est la plus utilisée.

```
SELECT Nom AS [Nom du Kart]
FROM Courses.Kart K
LEFT JOIN Courses.Participation P
ON K.KartID = P.KartID
WHERE P.KartID IS NULL
```



Nom du Kart
Blue Falcon
Yoshimoto

Par contre, l'utilisation de la clause EXISTS/NOT EXISTS est plus générique et permet de répondre à bien d'autres questions.



❖ Any

L'opérateur **ANY** est très similaire à **EXISTS**. Il permet de **vérifier si une condition est vraie pour au moins une valeur** parmi les valeurs obtenues dans une sous-requête. L'opérateur avant **ANY** peut être **=, !=, <, >, <=** ou **>=**

```
SELECT Nom AS [Nom du Kart]  
FROM Courses.Kart K  
WHERE KartID = ANY ( SELECT KartID  
                     FROM Courses.Participation P  
                     WHERE P.KartID = K.KartID)
```



Nom
Kart Standard
Chabriolet
Kart Rétro
Koopa-mobile
Autorhino
Scootinette
Quad Wiggler
Quad Standard

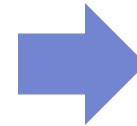
On sélectionne tous les noms de karts pour lesquels le KartID est égal à au moins un KartID dans la table Participation.



❖ All

L'opérateur **ALL** est similaire à **ANY** et permet de vérifier si une condition est vraie **pour toutes** les valeurs obtenues dans une sous-requête.

```
SELECT Nom AS [Nom du Kart]  
FROM Courses.Kart K  
WHERE Vitesse >= ALL ( SELECT Vitesse  
                        FROM Courses.Kart  
                        WHERE Poids > 3.0)
```



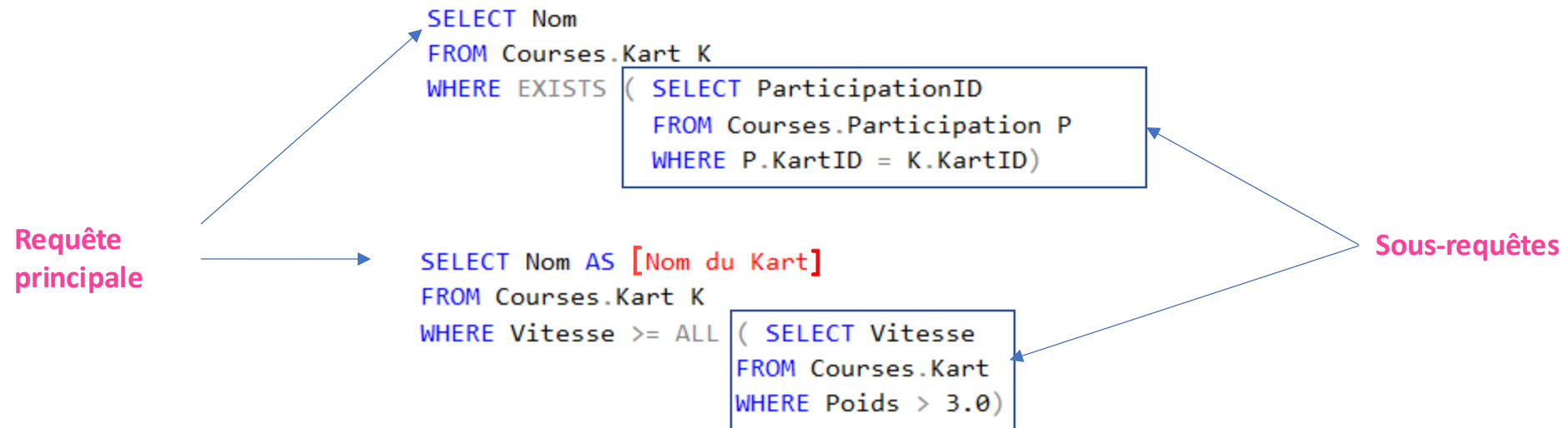
Nom
Blue Falcon

On sélectionne le nom des Kart pour lesquels la vitesse est plus grande ou égale à toutes les Vitesses dans la table Kart pour les Karts ayant un Poids supérieur à 3.0.



❖ Sous-requêtes

Nous avons utilisé beaucoup de sous-requêtes sans en parler plus que cela.
On en avait besoin pour les opérateurs EXISTS, ANY, ALL



Les sous-requêtes sont exécutées AVANT la requête principale



❖ Sous-requêtes

L'utilisation la plus fréquente des sous-requêtes est pour trouver les rangées d'une table dont une valeur est $>$, $>=$, $<$, $<=$ à la moyenne des valeurs de toutes les rangées de la table.

Requête
principale

```
SELECT Nom AS [Nom du Kart]  
FROM Courses.Kart K  
WHERE Vitesse >= (  
    SELECT AVG(Vitesse)  
    FROM Courses.Kart )
```

Sous-requête

La sous-requête, exécutée en premier, nous donne la vitesse moyenne de tous les karts de la table Kart.

La requête principale nous donne le nom de chaque Kart dont la vitesse est supérieure à la valeur donnée par la sous-requête.



❖ Fonctions et opérations sur colonnes

◆ Cette section ne sera pas exhaustive, mais elle contient de nombreuses fonctions intéressantes à utiliser sur les colonnes.

- Opérateurs arithmétiques
- Concaténation
- Chaînes de caractères
- Dates
- Formatage



❖ Fonctions et opérations sur colonnes

◆ Opérateurs arithmétiques

```
SELECT Colonne1 * Colonne2 AS Nom, Colonne3 FROM Table1
```

On peut faire des opérations arithmétiques sur les colonnes. (+, -, *, /, %)

```
SELECT Nom, ( (Acceleration * TenueDeRoute) / 36 ) AS [Confort de conduite (%)]  
FROM Courses.Kart
```



Nom	Confort de conduite (%)
Kart Standard	0.2500000
Chabriolet	0.1736111
Kart Rétro	0.4444444
Koopa-mobile	0.1562500
Blue Falcon	0.0729166
Autorhino	0.5312500
Scootinette	0.7638888
Yoshimoto	0.1718750
Quad Wiggler	0.4166666
Quad Standard	0.2777777

Attention ! La division de deux entiers donne un entier !

Si on veut diviser deux variables entières et avoir un résultat avec virgule, il faut :

- Soit faire un cast d'une des variables
 - Soit multiplier une des variables par 1.0
 - Le calcul ci-dessus donne des décimales parce que au moins Acceleration ou TenueDeRoute est un champ de type Float ou Decimal.
- Si les deux champs Acceleration et TenueDeRoute étaient des entiers, vous pourriez diviser par 36.0 pour obtenir un résultat avec des décimales.



❖ Fonctions et opérations sur colonnes

◆ Concaténation

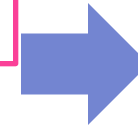
```
SELECT Colonne1 + Colonne2 AS [Nom], Colonne3 FROM Table1
```

Si les colonnes contiennent des chaînes de caractères, l'opérateur + permettra de faire une concaténation. Attention ! La concaténation avec l'opérateur + retourne NULL dès qu'une des colonnes contient la valeur NULL.

```
SELECT CONCAT(Colonne1, Colonne2, ...) AS [Nom], Colonne3 FROM Table1
```

La fonction **CONCAT(...)** permet de concaténer deux colonnes ou plus de plusieurs types. Cette fois-ci, si une colonne vaut NULL, elle sera convertie en chaîne vide. **CONCAT()** retourne seulement NULL si **TOUTES** les colonnes sont NULL.

```
SELECT CONCAT('Le kart ', Nom, ' a ', Vitesse) AS [Description]  
FROM Courses.Kart
```



Description
Le kart Rétro a 2.75
Le kart Proto 8 a 3.75
Le kart Chabriolet a 3.25
Le kart Beat-bolide a 3.00
Le kart Paracoccinelly a 2.25
Le kart Caravéloce a 2.50



❖ Fonctions et opérations sur colonnes

◆ Fonctions de chaînes

```
SELECT LEFT(Colonne1, x), ... FROM Table1
```

LEFT() conserve seulement les x premiers caractères d'une chaîne. **RIGHT()** conserve seulement les x derniers caractères d'une chaîne.

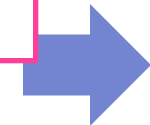
```
SELECT SUBSTRING(Colonne1, x, y), ... FROM Table1
```

SUBSTRING() conserve seulement les y premiers caractères d'une chaîne à partir du caractère à la position X. Attention : le premier caractère d'une chaîne est à la position 1. (Et non à la position 0)

```
SELECT TRIM(Colonne1), ... FROM Table1
```

TRIM() retire les « leading spaces » et les « trailing spaces » (espaces au début et à la fin d'un string).

```
SELECT SUBSTRING(Nom, 1, 6) AS [DebutNomKarts]  
FROM Courses.Kart
```



DebutNomKarts
Kart S
Chabri
Kart R
Koopa-
Blue F
Autorh
Scooti
Yoshim
Quad W
Quad S



❖ Fonctions et opérations sur colonnes

◆ Fonctions de chaînes

```
SELECT UPPER(Colonne1 + 'allo'), ... FROM Table1
```

UPPER() transforme tous les caractères alphabétiques en majuscules dans une chaîne.

```
SELECT LOWER(Colonne1 + ' ' + Colonne2), ... FROM Table1
```

LOWER() transforme tous les caractères alphabétiques en minuscules dans une chaîne.

```
SELECT LEN(Colonne1), ... FROM Table1
```

LEN() retourne la taille d'une chaîne.

```
SELECT UPPER(Nom) AS [Nom en Majuscules], LEN(Nom) AS [Nb de caractères]  
FROM Courses.Kart
```



Nom en Majuscules	Nb de caractères
AUTORHINO	9
BEAT-BOLIDE	11
BLUE FALCON	11
CARAVÉLOCE	10
CHABRIOLET	10
PARACOCINELLY	14
PROPULSAR	9



❖ Fonctions et opérations sur colonnes

◆ Fonctions de dates

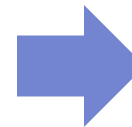
```
SELECT GETDATE() AS [Date], ... FROM Table1
```

GETDATE() retourne la date actuelle dans le format **datetime** 'AAAA-MM-DD HH:MI:SS.NNN'

```
SELECT DAY(Col2) AS [Jour], MONTH(Col2) AS [Mois], YEAR(Col2) AS [Année],  
... FROM Table1
```

DAY(), MONTH() et YEAR() retournent la valeur du jour / mois / année sous forme de nombre entier lorsqu'on leur passe une date (**datetime**, **date**, **varchar** bien formaté, etc.) en paramètre.

```
SELECT GETDATE() AS [Date], DAY(GETDATE()) AS [Jour], MONTH(GETDATE()) AS [Mois]
```



Date	Jour	Mois
2023-01-31 18:25:12.817	31	1



❖ Fonctions et opérations sur colonnes

◆ Fonctions de dates

```
SELECT DATEDIFF(x, Date1, Date2) AS [...], ... FROM Table1
```

DATEDIFF() retourne la différence entre deux dates. (Date2 - Date1) Le résultat est un entier et l'unité pour exprimer la différence est choisie avec le paramètre x. (MONTH, DAY, YEAR, HOUR, WEEK, etc.).

```
SELECT DATEADD( x, Date1 , y) AS [...], .. FROM Table1
```

DATEADD() retourne une date à laquelle on a ajouté y jours, mois, années...selon le paramètre x choisi. Le résultat est une date et l'unité y ajoute x (MONTH, DAY, YEAR, HOUR, WEEK, etc.) à Date1.

```
SELECT DATEDIFF(DAY, '20200101', GETDATE()) AS [Différence en jours]
```



Différence en jours
1138

```
SELECT DATEADD( day, DateCommande, 2 ) AS [Date de livraison]  
FROM dbo.Commande
```



La livraison sera 2 jours après la date de la commande.



❖ Fonctions et opérations sur colonnes

◆ Fonctions de dates

```
SELECT CAST(DateDeCommande AS [DATE]) AS [Date de commande], ... FROM Table1
```

CAST() permet d'afficher seulement la portion de la date d'un champ de type datetime, sans les heures, minutes, sec...

```
SELECT TOP(5) CAST(DatePartie AS [DATE]) AS [Date de la partie]  
FROM Courses.Participation  
ORDER BY DatePartie DESC
```

Afficher les dates des 5 dernières parties qui ont été faites, sans la portion heures...



Résultats		Message
	Date de la partie	
1	2022-12-28	
2	2022-12-27	
3	2022-12-26	
4	2022-12-25	
5	2022-12-24	



❖ Fonctions et opérations sur colonnes

◆ Formatage

- La fonction `FORMAT()` est extrêmement polyvalente.

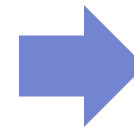
```
SELECT FORMAT(314.591, '.##') AS [Nom], ... FROM Table1
```

`FORMAT()` peut transformer des valeurs numériques avec un format qui spécifie le nombre de chiffres après la virgule.

```
SELECT FORMAT(314.591, 'N2', 'en-US') AS [Nom], ... FROM Table1
```

Avec des formats comme `'N1'` ou `'N2'`, on peut spécifier qu'il y a 1 ou 2 chiffres décimaux après la virgule. Il peut y avoir un troisième paramètre pour spécifier la *culture*. Par exemple, avec `'en-US'`, les décimales seront précédées d'un point, alors qu'avec `'fr-be'`, les décimales auraient été précédées d'une virgule.

```
SELECT FORMAT(123456.654321, '.###') AS [Nombre],  
FORMAT(123456.654321, 'N2', 'en-US') AS [Nombre_US]
```



Nombre	Nombre_US
123456.654	123,456.65



❖ Fonctions et opérations sur colonnes

◆ Formatage

- La fonction `FORMAT()` est extrêmement polyvalente.

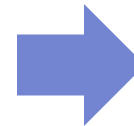
```
SELECT FORMAT(155.21, 'C', 'fr-fr') AS [Nom], ... FROM Table1
```

Avec le format `'C'`, la valeur numérique sera transformée en somme monétaire. Selon la valeur du paramètre culture, le montant sera affiché différemment. Ex : Avec `'fr-fr'`, on affiche € à droite du montant, alors qu'avec `'en-US'`, on affiche \$ à gauche du montant.

```
SELECT FORMAT(GETDATE(), 'd', 'en-US') AS [Nom], ... FROM Table1
```

Avec le format `'d'`, on peut afficher une date de manière cohérente avec la culture passée en paramètre. Par exemple, pour le 12 janvier 2012, nous aurons `'01/12/2012'` pour `'en-US'`, alors que nous aurions `'12/01/2012'` pour `'fr-fr'` ou `'12.01.2012'` pour `'ru-ru'`.

```
SELECT FORMAT(12345.67, 'C', 'jp-jp') AS [Monnaie japonaise],  
FORMAT(GETDATE(), 'd', 'en-US') AS [Date américaine]
```



Monnaie japonaise	Date américaine
¥12,345.67	1/31/2023