

A close-up photograph of a bulk candy store. Numerous white plastic trays are filled with various types of wrapped candies in different colors like red, blue, yellow, and orange. A person's hand, wearing a blue denim sleeve, is reaching into one of the trays to pick up a candy. In the foreground, there are several boxes of Trident gum, some in their original packaging and others unpacked into the trays.

Module 03 - DML

Objectifs – Révisions

- Révision des quatre (4) opérations classiques :
 - SELECT :
 - INSERT
 - UPDATE
 - DELETE

SELECT – Structure

- Forme générale

```
[ WITH <common table expression> [ ,...n ] ]  
SELECT <select_criteria>  
[;]  
  
<select_criteria> ::=  
[ TOP ( top_expression ) ]  
[ ALL | DISTINCT ]  
{ * | column_name | expression } [ ,...n ]  
[ FROM { table_source } [ ,...n ] ]  
[ WHERE <search_condition> ]  
[ GROUP BY <group_by_clause> ]  
[ HAVING <search_condition> ]  
[ ORDER BY <order_by_expression> ]  
[ OPTION ( <query_option> [ ,...n ] ) ]
```

SELECT : projection

TOP : nombre limite d'enregistrements

FROM : tables sources avec possibles jointures => D1

WHERE : filtre appliqué sur l'ensemble de données D1 issues du FROM => D2

GROUP BY : critères pour former des groupes à partir de D2 => D3

HAVING : similaire au WHERE mais sur les données D3 => D4

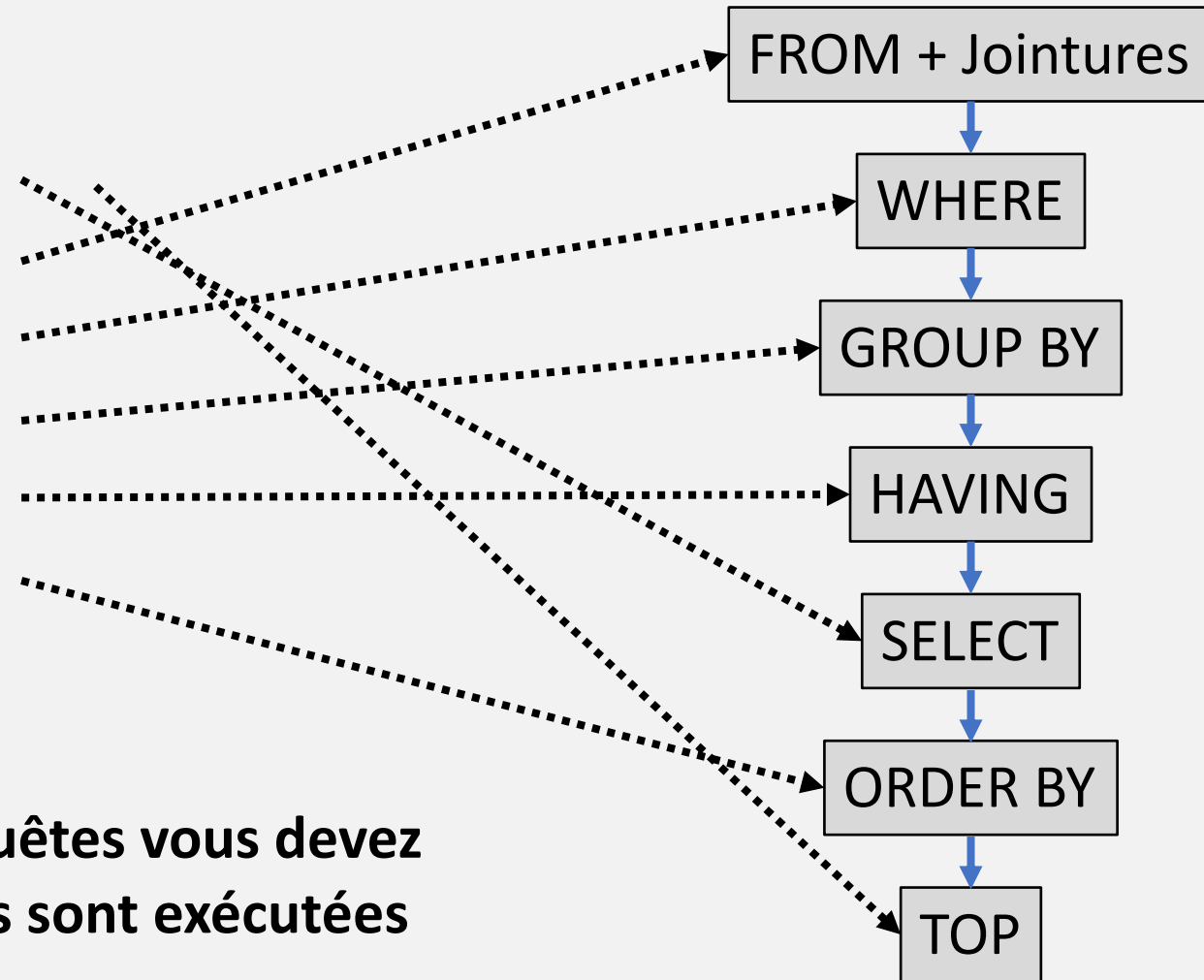
ORDER BY : critères de tri sur les données D4 => D5

SELECT – Différence entre déclaration et exécution

- Déclaration

- SELECT [TOP] ...
- FROM + Jointures
- WHERE ...
- GROUP BY ...
- HAVING ...
- ORDER BY ...

**=> En écrivant vos requêtes vous devez
pense à comment elles sont exécutées**



(Version simplifiée)

SELECT – Différence entre déclaration et exécution

SQL	Linq (C#)	JavaScript
SELECT	.Select(lambda)	.Map(lambda)
FROM	<collection>	<array>
JOIN	.Join(lambda)	.foreach + .filter
WHERE/HAVING	.Where(prédicat)	.filter(prédicat)
GROUP BY	.GroupBy(lambda)	.reduce(lambda, {}) .group(lambda)
ORDER BY	.OrderBy(lambda)	.sort(lambda)

```
var groupBy = function(xs, key) {  
  return xs.reduce(function(rv, x) {  
    (rv[x[key]] = rv[x[key]] || []).push(x);  
    return rv;  
  }, {});  
};  
  
console.log(groupBy(['one', 'two', 'three'], 'length'));  
  
// => {"3": ["one", "two"], "5": ["three"]}
```

Exemple 1 – Exécution – SELECT simple

SELECT nom **FROM** chien **WHERE** nom **LIKE** 'A%'

BD

Chien

Proprietaire

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
2	Marieke	1		2	Léopoldine	
3	Médor	2		3	Sonia	
4	Abel	1				
5	Maya	3				
6	Max	2				
7	Scotch	NULL				

FROM chien

id	nom	proprietaireId	...
1	Ada	1	
2	Marieke	1	
3	Médor	2	
4	Abel	1	
5	Maya	3	
6	Max	2	
7	Scotch	NULL	

Exemple 1 – Exécution – SELECT simple

SELECT nom **FROM** chien **WHERE** nom **LIKE** 'A%'

id	nom	proprietaireId	...
1	Ada	1	
2	Marieke	1	
3	Médor	2	
4	Abel	1	
5	Maya	3	
6	Max	2	
7	Scotch	NULL	

WHERE nom **LIKE** 'A%'

id	nom	proprietaireId	...
1	Ada	1	
4	Abel	1	

Exemple 1 – Exécution – SELECT simple

```
SELECT nom FROM chien WHERE nom LIKE 'A%'
```

id	nom	proprietaireId	...
1	Ada	1	
4	Abel	1	

SELECT nom

nom
Ada
Abel

Exemple 2 – Exécution – INNER JOIN

```
SELECT nom AS nomChien, prenom AS prenomProprietaire
FROM chien c
INNER JOIN proprietaire p ON c.proprietaireId = p.id
```

BD

Chien

Proprietaire

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
2	Marieke	1		2	Léopoldine	
3	Médor	2		3	Sonia	
4	Abel	1				
5	Maya	3				
6	Max	2				
7	Scotch	NULL				

FROM chien c
INNER JOIN proprietaire p

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
1	Ada	1		2	Léopoldine	
1	Ada	1		3	Sonia	
2	Marieke	1		1	Hortense	
2	Marieke	1		2	Léopoldine	
2	Marieke	1		3	Sonia	
3	Médor	2		1	Hortense	
3	Médor	2		2	Léopoldine	
3	Médor	2		3	Sonia	
4	Abel	1		1	Hortense	
4	Abel	1		2	Léopoldine	
4	Abel	1		3	Sonia	
5	Maya	3		1	Hortense	
5	Maya	3		2	Léopoldine	
5	Maya	3		3	Sonia	
6	Max	2		1	Hortense	
6	Max	2		2	Léopoldine	
6	Max	2		3	Sonia	

Exemple 2 – Exécution – INNER JOIN

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
1	Ada	1		2	Léopoldine	
1	Ada	1		3	Sonia	
2	Marieke	1		1	Hortense	
2	Marieke	1		2	Léopoldine	
2	Marieke	1		3	Sonia	
3	Médor	2		1	Hortense	
3	Médor	2		2	Léopoldine	
3	Médor	2		3	Sonia	
4	Abel	1		1	Hortense	
4	Abel	1		2	Léopoldine	
4	Abel	1		3	Sonia	
5	Maya	3		1	Hortense	
5	Maya	3		2	Léopoldine	
5	Maya	3		3	Sonia	
6	Max	2		1	Hortense	
6	Max	2		2	Léopoldine	
6	Max	2		3	Sonia	

```
SELECT nom AS nomChien, prenom AS prenomProprietaire
FROM chien c
INNER JOIN proprietaire p ON c.proprietaireId = p.id
```

ON c.proprietaireId = p.id →

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
2	Marieke	1		1	Hortense	
3	Médor	2		2	Léopoldine	
4	Abel	1		1	Hortense	
5	Maya	3		3	Sonia	
6	Max	2		2	Léopoldine	

Exemple 2 – Exécution – INNER JOIN

```
SELECT nom AS nomChien, prenom AS prenomProprietaire
FROM chien c
INNER JOIN proprietaire p ON c.proprietaireId = p.id
```

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
2	Marieke	1		1	Hortense	
3	Médor	2		2	Léopoldine	
4	Abel	1		1	Hortense	
5	Maya	3		3	Sonia	
6	Max	2		2	Léopoldine	

```
SELECT nom AS nomChien,
prenom AS prenomProprietaire
```

nomChien	prenomProprietaire
Ada	Hortense
Marieke	Hortense
Médor	Léopoldine
Abel	Hortense
Maya	Sonia
Max	Léopoldine

Exemple 3 – Exécution – OUTER JOIN

SELECT nom **AS** nomChien, prenom **AS** prenomProprietaire
FROM chien c
LEFT JOIN proprietaire p **ON** c.proprietaireId = p.id

BD

Chien

Proprietaire

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
2	Marieke	1		2	Léopoldine	
3	Médor	2		3	Sonia	
4	Abel	1				
5	Maya	3				
6	Max	2				
7	Scotch	NULL				

FROM chien c
LEFT JOIN proprietaire p

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
1	Ada	1		2	Léopoldine	
1	Ada	1		3	Sonia	
2	Marieke	1		1	Hortense	
2	Marieke	1		2	Léopoldine	
2	Marieke	1		3	Sonia	
3	Médor	2		1	Hortense	
3	Médor	2		2	Léopoldine	
3	Médor	2		3	Sonia	
4	Abel	1		1	Hortense	
4	Abel	1		2	Léopoldine	
4	Abel	1		3	Sonia	
5	Maya	3		1	Hortense	
5	Maya	3		2	Léopoldine	
5	Maya	3		3	Sonia	
6	Max	2		1	Hortense	
6	Max	2		2	Léopoldine	
6	Max	2		3	Sonia	
7	Scotch	NULL		NULL	NULL	NULL

Exemple 3 – Exécution – OUTER JOIN

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
1	Ada	1		2	Léopoldine	
1	Ada	1		3	Sonia	
2	Marieke	1		1	Hortense	
2	Marieke	1		2	Léopoldine	
2	Marieke	1		3	Sonia	
3	Médor	2		1	Hortense	
3	Médor	2		2	Léopoldine	
3	Médor	2		3	Sonia	
4	Abel	1		1	Hortense	
4	Abel	1		2	Léopoldine	
4	Abel	1		3	Sonia	
5	Maya	3		1	Hortense	
5	Maya	3		2	Léopoldine	
5	Maya	3		3	Sonia	
6	Max	2		1	Hortense	
6	Max	2		2	Léopoldine	
6	Max	2		3	Sonia	
7	Scotch	NULL		NULL	NULL	NULL

```
SELECT nom AS nomChien, prenom AS prenomProprietaire
FROM chien c
LEFT JOIN proprietaire p ON c.proprietaireId = p.id
```

ON c.proprietaireId = p.id

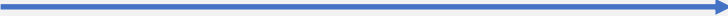
id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
2	Marieke	1		1	Hortense	
3	Médor	2		2	Léopoldine	
4	Abel	1		1	Hortense	
5	Maya	3		3	Sonia	
6	Max	2		2	Léopoldine	
7	Scotch	NULL		NULL	NULL	NULL

Exemple 3 – Exécution – OUTER JOIN

```
SELECT nom AS nomChien, prenom AS prenomProprietaire
FROM chien c
LEFT JOIN proprietaire p ON c.proprietaireId = p.id
```

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
2	Marieke	1		1	Hortense	
3	Médor	2		2	Léopoldine	
4	Abel	1		1	Hortense	
5	Maya	3		3	Sonia	
6	Max	2		2	Léopoldine	
7	Scotch	NULL		NULL	NULL	NULL

SELECT nom AS nomChien,
prenom AS prenomProprietaire



nomChien	prenomProprietaire
Ada	Hortense
Marieke	Hortense
Médor	Léopoldine
Abel	Hortense
Maya	Sonia
Max	Léopoldine
Scotch	NULL

Exemple 4 – Exécution – + HAVING

```
SELECT prenom AS prenomProprietaire, COUNT(*) AS nombreChiens
```

```
FROM chien c
```

```
INNER JOIN proprietaire p ON c.proprietaireId = p.id
```

```
WHERE c.nom NOT LIKE '%i%'
```

```
GROUP BY p.id, p.prenom
```

```
HAVING COUNT(*) > 1
```

```
ORDER BY prenomProprietaire DESC
```

BD

Chien

Proprietaire

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
2	Marieke	1		2	Léopoldine	
3	Médor	2		3	Sonia	
4	Abel	1				
5	Maya	3				
6	Max	2				
7	Scotch	NULL				

```
FROM chien c  
INNER JOIN proprietaire p
```

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
1	Ada	1		2	Léopoldine	
1	Ada	1		3	Sonia	
2	Marieke	1		1	Hortense	
2	Marieke	1		2	Léopoldine	
2	Marieke	1		3	Sonia	
3	Médor	2		1	Hortense	
3	Médor	2		2	Léopoldine	
3	Médor	2		3	Sonia	
4	Abel	1		1	Hortense	
4	Abel	1		2	Léopoldine	
4	Abel	1		3	Sonia	
5	Maya	3		1	Hortense	
5	Maya	3		2	Léopoldine	
5	Maya	3		3	Sonia	
6	Max	2		1	Hortense	
6	Max	2		2	Léopoldine	
6	Max	2		3	Sonia	

Exemple 4 – Exécution – + HAVING

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
1	Ada	1		2	Léopoldine	
1	Ada	1		3	Sonia	
2	Marieke	1		1	Hortense	
2	Marieke	1		2	Léopoldine	
2	Marieke	1		3	Sonia	
3	Médor	2		1	Hortense	
3	Médor	2		2	Léopoldine	
3	Médor	2		3	Sonia	
4	Abel	1		1	Hortense	
4	Abel	1		2	Léopoldine	
4	Abel	1		3	Sonia	
5	Maya	3		1	Hortense	
5	Maya	3		2	Léopoldine	
5	Maya	3		3	Sonia	
6	Max	2		1	Hortense	
6	Max	2		2	Léopoldine	
6	Max	2		3	Sonia	

```
SELECT prenom AS prenomProprietaire, COUNT(*) AS nombreChiens
FROM chien c
INNER JOIN proprietaire p ON c.proprietaireId = p.id
WHERE c.nom NOT LIKE '%i%'
GROUP BY p.id, p.prenom
HAVING COUNT(*) > 1
ORDER BY prenomProprietaire DESC
```

ON c.proprietaireId = p.id



id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
2	Marieke	1		1	Hortense	
3	Médor	2		2	Léopoldine	
4	Abel	1		1	Hortense	
5	Maya	3		3	Sonia	
6	Max	2		2	Léopoldine	

Exemple 4 – Exécution – + HAVING

```
SELECT prenom AS prenomProprietaire, COUNT(*) AS nombreChiens
FROM chien c
INNER JOIN proprietaire p ON c.proprietaireId = p.id
WHERE c.nom NOT LIKE '%i%'
GROUP BY p.id, p.prenom
HAVING COUNT(*) > 1
ORDER BY prenomProprietaire DESC
```

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
2	Marieke	1		1	Hortense	
3	Médor	2		2	Léopoldine	
4	Abel	1		1	Hortense	
5	Maya	3		3	Sonia	
6	Max	2		2	Léopoldine	

WHERE c.nom NOT LIKE '%i%'

id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
3	Médor	2		2	Léopoldine	
4	Abel	1		1	Hortense	
5	Maya	3		3	Sonia	
6	Max	2		2	Léopoldine	

Exemple 4 – Exécution – + HAVING

```
SELECT prenom AS prenomProprietaire, COUNT(*) AS nombreChiens
FROM chien c
INNER JOIN proprietaire p ON c.proprietaireId = p.id
WHERE c.nom NOT LIKE '%i%'
GROUP BY p.id, p.prenom
HAVING COUNT(*) > 1
ORDER BY prenomProprietaire DESC
```

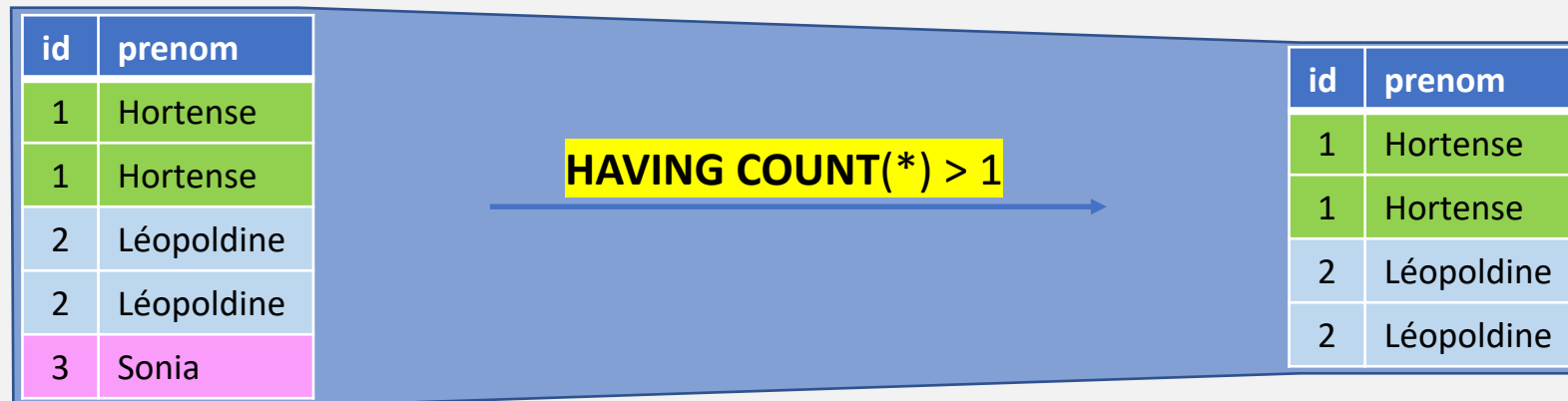
id	nom	proprietaireId	...	id	prenom	...
1	Ada	1		1	Hortense	
3	Médor	2		2	Léopoldine	
4	Abel	1		1	Hortense	
5	Maya	3		3	Sonia	
6	Max	2		2	Léopoldine	

GROUP BY p.id, p.prenom

id	prenom
1	Hortense
1	Hortense
2	Léopoldine
2	Léopoldine
3	Sonia

Exemple 4 – Exécution – + HAVING

```
SELECT prenom AS prenomProprietaire, COUNT(*) AS nombreChiens
FROM chien c
INNER JOIN proprietaire p ON c.proprietaireId = p.id
WHERE c.nom NOT LIKE '%i%'
GROUP BY p.id, p.prenom
HAVING COUNT(*) > 1
ORDER BY prenomProprietaire DESC
```

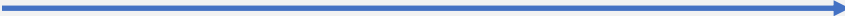


Exemple 4 – Exécution – + HAVING

```
SELECT prenom AS prenomProprietaire, COUNT(*) AS nombreChiens
FROM chien c
INNER JOIN proprietaire p ON c.proprietaireId = p.id
WHERE c.nom NOT LIKE '%i%'
GROUP BY p.id, p.prenom
HAVING COUNT(*) > 1
ORDER BY prenomProprietaire DESC
```

id	prenom
1	Hortense
1	Hortense
2	Léopoldine
2	Léopoldine

```
SELECT prenom AS prenomProprietaire,
COUNT(*) AS nombreChiens
```



prenomProprietaire	nombreChiens
Hortense	2
Léopoldine	2

Exemple 4 – Exécution – + HAVING

```
SELECT prenom AS prenomProprietaire, COUNT(*) AS nombreChiens
FROM chien c
INNER JOIN proprietaire p ON c.proprietaireId = p.id
WHERE c.nom NOT LIKE '%i%'
GROUP BY p.id, p.prenom
HAVING COUNT(*) > 1
ORDER BY prenomProprietaire DESC
```

prenomProprietaire	nombreChiens
Hortense	2
Léopoldine	2

ORDER BY prenomProprietaire DESC



prenomProprietaire	nombreChiens
Léopoldine	2
Hortense	2

Résumé SELECT

(Représentation pouvant être demandée en exercices ou examen)

```
SELECT c.proprietaireId, COUNT(*) AS nombreChiens
FROM chien c
WHERE c.nom NOT LIKE '%i%'
GROUP BY c.proprietaireId
HAVING COUNT(*) > 1
ORDER BY c.proprietaireId DESC
```

- (5)
- (1)
- (2)
- (3)
- (4)
- (6)

(1) FROM chien c

id	nom	proprietaireId	...
1	Ada	1	
2	Marieke	1	
3	Médor	2	
4	Abel	1	
5	Maya	3	
6	Max	2	
7	Scotch	NULL	

(2) WHERE c.nom NOT LIKE '%i%'

id	nom	proprietaireId	...
1	Ada	1	
2	Marieke	1	
3	Médor	2	
4	Abel	1	
5	Maya	3	
6	Max	2	
7	Scotch	NULL	

(3) GROUP BY c.proprietaireId

id	nom	proprietaireId	...
1	Ada	1	
4	Abel	1	
3	Médor	2	
6	Max	2	
5	Maya	3	
7	Scotch	NULL	

(4) HAVING COUNT(*) > 1

id	nom	proprietaireId	...
1	Ada	1	
4	Abel	1	
3	Médor	2	
6	Max	2	

(5) SELECT c.proprietaireId, COUNT(*) AS nombreChiens

proprietaireId	nombreChiens
1	2
2	2

(6) ORDER BY c.proprietaireId DESC

proprietaireId	nombreChiens
2	2
1	2

Insertion

```
[ WITH <common_table_expression> [ ,...n ] ]
INSERT
{
    [ TOP ( expression ) [ PERCENT ] ]
    [ INTO ]
    { <object> | rowset_function_limited
      [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
    }
    {
        [ ( column_list ) ]
        [ <OUTPUT Clause> ]
        { VALUES ( { DEFAULT | NULL | expression } [ ,...n ] ) [ ,...n ]
          | derived_table
          | execute_statement
          | <dml_table_source>
          | DEFAULT VALUES
        }
    }
}
[;]
```

```
<object> ::=
{
    [ server_name . database_name . schema_name .
      | database_name . [ schema_name ] .
      | schema_name .
    ]
    table_or_view_name
}
```

```
<dml_table_source> ::=
    SELECT <select_list>
    FROM ( <dml_statement_with_output_clause> )
        [ AS ] table_alias [ ( column_alias [ ,...n ] ) ]
    [ WHERE <search_condition> ]
    [ OPTION ( <query_hint> [ ,...n ] ) ]
```

Mise à jour

```
[ WITH <common_table_expression> [...n] ]
UPDATE
  [ TOP ( expression ) [ PERCENT ] ]
  { { table_alias | <object> | rowset_function_limited
    [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
  }
  | @table_variable
}
SET
  { column_name = { expression | DEFAULT | NULL }
    | { udt_column_name. { { property_name = expression
                          | field_name = expression }
                        | method_name ( argument [ ,...n ] )
                      }
    }
    | column_name { .WRITE ( expression , @Offset , @Length ) }
    | @variable = expression
    | @variable = column = expression
    | column_name { += | -= | *= | /= | %= | &= | ^= | |= } expression
    | @variable { += | -= | *= | /= | %= | &= | ^= | |= } expression
    | @variable = column { += | -= | *= | /= | %= | &= | ^= | |= } expression
  } [ ,...n ]

[ <OUTPUT Clause> ]
[ FROM { <table_source> } [ ,...n ] ]
```

```
[ WHERE { <search_condition>
          | { [ CURRENT OF
              { { [ GLOBAL ] cursor_name }
                | cursor_variable_name
              }
            ]
          }
        ]
[ OPTION ( <query_hint> [ ,...n ] ) ]
[ ; ]

<object> ::=
{
  [ server_name . database_name . schema_name .
    | database_name . [ schema_name ] .
    | schema_name .
  ]
  table_or_view_name}
```


Suppression

```
[ WITH <common_table_expression> [ ,...n ] ]
DELETE
[ TOP ( expression ) [ PERCENT ] ]
[ FROM ]
{ { table alias
  | <object>
  | rowset_function_limited
  [ WITH ( table_hint_limited [ ...n ] ) ] }
  | @table_variable
}
[ <OUTPUT Clause> ]
[ FROM table source [ ,...n ] ]
[ WHERE { <search_condition>
          | { [ CURRENT OF
              { { [ GLOBAL ] cursor_name }
                | cursor_variable_name
              }
            ]
          }
        ]
      ]
[ OPTION ( <Query Hint> [ ,...n ] ) ]
[; ]
```

```
<object> ::=
{
    [ server_name.database_name.schema_name.
      | database_name. [ schema_name ] .
      | schema_name.
    ]
    table_or_view_name
}
```

Construction des requêtes - NULL

- Assurez-vous de la signification de NULL dans votre modèle :
 - Valeur inconnue **OU** oubliée ?
 - Personne.NAS **IS NULL** : la personne n'a pas de NAS **OU** on ne le connaît pas
 - Une chaîne vide "" ou la valeur 0 **ne sont pas nulles**
- Attention aux comparaisons avec NULL :
 - `NULL = NULL => FALSE`
 - `NULL <> NULL => FALSE`
- Pour effectuer des comparaisons, utilisez « IS NULL » ou « IS NOT NULL »



0 vs NULL

Construction des requêtes - NULL

- Attention aux opérateurs avec NULL :
 - $1 + \text{NULL} \Rightarrow \text{NULL}$
 - $\text{'ABC'} + \text{NULL} \Rightarrow \text{NULL}$
- Si une valeur peut être nulle, utilisez une des fonctions **COALESCE** (plusieurs valeurs) ou **ISNULL** (deux valeurs) qui renvoient la première valeur non nulle :

```
SELECT 1 + COALESCE(NULL, NULL, 2) + ISNULL(NULL, 4)
```

Précédence des opérateurs

- Précédence des opérateurs :
 - Opérateurs arithmétiques
 - Opérateurs logiques :
 - **NOT**
 - **AND**
 - **OR**
- On peut modifier cette ordre en utilisant des parenthèses
- Ajoutez des parenthèses, même si elles sont inutiles, limite les ambiguïtés et rend le code plus lisible

Conseils – SELECT

- Toujours valider vos données
- Comprendre la signification de **NULL**
- Créez des alias pour les tables afin de mieux les référer
- Utiliser seulement des « **INNER JOIN** » et des « **LEFT JOIN** » : oubliez les « **RIGHT JOIN** »
- Assurez vous que les colonnes de jointure sont uniques
- Si vous utilisez un « **GROUP BY** », les valeurs du « **SELECT** » doivent être :
 - Soit calculées à partir des colonnes décrites dans le « **GROUP BY** »
 - Soit calculées par une fonction d'agrégat (Ex. : **COUNT**, **MIN**, **MAX**, **SUM**, **AVG**, etc.)

Conseils – INSERT

- **(Démarrez une transaction** : vue plus loin dans le cours)
- Si vous n'écrivez pas un script de création / insertion de données de base : encadrez vos inserts par une alternative tout le temps fausse (**IF 1=0 BEGIN [...] END**);
- Indiquez le nom des colonnes lors de l'insertion (L'ordre ou le nombre de colonnes pourraient changer avec le temps)



Conseils – DELETE / UPDATE

- **(Démarrez une transaction** : vue plus loin dans le cours)
- Pour ne pas exécuter les commandes par erreur : encadrez vos inserts par une alternative tout le temps fausse (**IF 1=0 BEGIN [...] END**);
- Écrivez en premier l’instruction **WHERE** avant d’écrire le reste de la commande
- Testez le filtre avec un **SELECT** avant d’exécuter le **DELETE / UPDATE**