



# Conception de base de données

## Langage d'interrogation de données de SQL

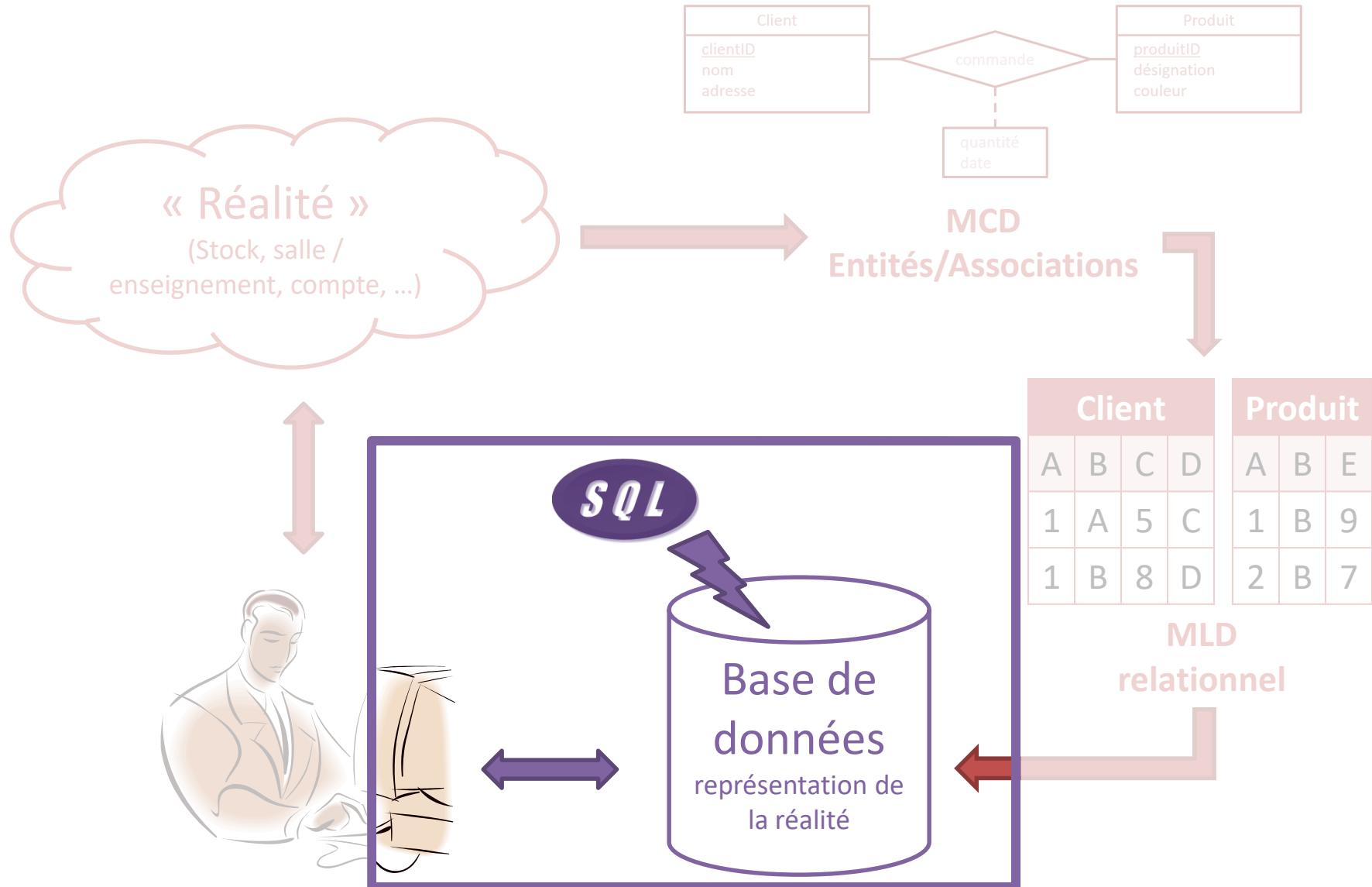
L3 Informatique

Antoine Spicher

[antoine.spicher@u-pec.fr](mailto:antoine.spicher@u-pec.fr)

---

# Big Picture



# Présentation rapide de SQL

## ■ *Structured Query Language*

- Origine
  - 1970, D. Chamberlain et R. Boyce, IBM
  - Inspiré du **modèle relationnel** de Codd
  - Nom originel : *Structured English QUERy Langage* (SEQUEL)
  - 1979, commercialisé par *Relational Software, Inc.* (*Oracle Corporation*)
- Normalisations
  - SQL-86 : recommandation de l'ANSI (American National Standards Institute)
  - **SQL-92** : révision majeure (ce cours développe une simplification de SQL-92)
  - 1999, 2003, 2008, 2011
- Présentation du langage guidée par la syntaxe
  - dont la sémantique correspond à l'**algèbre relationnelle multi-ensembliste**

# Plan



- Requêtes simples
- Compositions de requêtes
- Petit guide pratique

# Requête simple : syntaxe générale

## ■ Notations

### □ Style

- Mot clé : **noir/bordeaux gras**
- Identifiant, nom de variable : *orange italique*
- Non-terminal : **bleu**
- Constante, opération : **violet**
- Type SQL (cf. CM  $SR \Rightarrow SQL$ ) : **vert**

### □ Syntaxe des règles

- Production : **Non-terminal** ::= ... | ... | ...

au lieu de «  $N \rightarrow \dots | \dots | \dots$  »

- Option : [ ... ]

au lieu de «  $\dots N' \dots [ \dots ] \ N' \rightarrow \dots | \varepsilon$  »

- Sous-alternative : { ... | ... }

au lieu de «  $\dots N' \dots [ \dots ] \ N' \rightarrow \dots | \dots$  »

- Répétition : elt<sub>1</sub>, elt<sub>2</sub>, ...

au lieu de «  $\dots N' \dots [ \dots ] \ N' \rightarrow elt \mid N', elt$  »

# Requête simple : syntaxe générale

## ■ Grammaire des requêtes simples

Composition de *clauses*

- une projection (**SELECT**)
- une sélection (**WHERE**)
- une construction d'une table source (**FROM**)
- un groupement avec sélection (**GROUP BY ... HAVING ...**)
- un tri (**ORDER BY**)

```
RequeteSimple ::= SELECT Projection  
                  FROM Source  
                  [ WHERE Condition ]  
                  [ GROUP BY Groupement  
                    [ HAVING Condition ] ]  
                  [ ORDER BY Ordre ]
```

Source ::= *Table* | ...

# Requête simple : syntaxe générale

## ■ Grammaire des requêtes simples

Composition de *clauses*

- une projection (**SELECT**)
- une sélection (**WHERE**)
- une construction d'une table source (**FROM**)
- un groupement avec sélection (**GROUP BY ... HAVING ...**)
- un tri (**ORDER BY**)

```
RequeteSimple ::= SELECT Projection  
                  FROM Source  
                  [ WHERE Condition ]  
                  [ GROUP BY Groupement  
                    [ HAVING Condition ] ]  
                  [ ORDER BY Ordre ]
```

Source ::= *Table* | ...

# Requête simple : projection

## ■ Syntaxe de la clause **SELECT**

- Suite d'attributs : *Attribut<sub>1</sub>, Attribut<sub>2</sub>, ...*

*Ne conserver qu'un sous-ensemble des colonnes d'une table*

```
SELECT NCom, NVin, Quantite  
FROM Commande
```

Commande	<u>NCom</u>	<u>NClient</u>	<u>NVin</u>	Date	Quantité
1	5	4	27/04/12	25	
2	3	2	25/01/13	100	
3	2	2	14/08/09	80	
4	5	1	08/11/10	100	

	<u>NCom</u>	<u>NVin</u>	Quantité
1	4	25	
2	2	100	
3	2	80	
4	1	100	

$\tilde{\pi}_{NCom, NVin, Quantité}(\text{Commande})$

# Requête simple : projection

## ■ Syntaxe de la clause **SELECT**

- Absence de projection : \*

*Permet de ne pas spécifier de projection (la clause SELECT n'étant pas optionnelle)*

**SELECT \* FROM Commande**

Toutes les colonnes de la table  
*Commande* sont conservées

Commande	NCom	NClient	NVin	Date	Quantité
1	5	4	27/04/12	25	
2	3	2	25/01/13	100	
3	2	2	14/08/09	80	
4	5	1	08/11/10	100	

# Requête simple : projection

## ■ Syntaxe de la clause **SELECT**

- Mots clés **ALL** et **DISTINCT**

*Conserve (ALL) / ne conserve pas (DISTINCT) les doublons (ALL par défaut)*

```
SELECT ALL Viticulteur  
FROM Vin
```

 $\tilde{\pi}_{\text{Viticulteur}}(\text{Vin})$ 

Avec **ALL** (ou sans mention explicite)  
les doublons sont conservés

```
SELECT DISTINCT Viticulteur  
FROM Vin
```

 $\pi_{\text{Viticulteur}}(\text{Vin})$ 

Avec **DISTINCT**  
les doublons sont supprimés

	Viticulteur	Vin	N <sub>Vin</sub>	Cru	Millesime	Viticulteur	Région		Viticulteur
1			1	Saint-Emilion	2002	1	Bordeaux		1
3			2	Champagne	1996	3	Champagne		3
1			3	Pauillac	1992	1	Bordeaux		6
6			4	Chaplis	2007	6	Bourgogne		

# Requête simple : projection

## ■ Syntaxe de la clause **SELECT**

- Expressions et (re)nommage :  $\text{Exp}_1 [ \text{AS} \text{ Colonne}_1 ]$ ,  $\text{Exp}_2 [ \text{AS} \text{ Colonne}_2 ]$ , ...

- *Créer de nouvelles colonnes en combinant les valeurs des anciennes*

La spécification d'un nom pour identifier la colonne est souvent nécessaire

Viticulteur	<u>NUM</u>	Nom	Prénom	Région	CA
1	Barré	Henri	Bordeaux	100000	
3	Hermelin	Jean-Pierre	Champagne	150000	
4	Fort	Valérie	Languedoc	100000	
6	Schmidt	Thomas	Bourgogne	200000	



	NUM	
1	85000	
3	127500	
4	85000	
6	170000	

Prévoir pour chaque viticulteur un  
*nouveau chiffre d'affaire*  
*considérant une récession de 15%*

```
SELECT Num, CA * 85 / 100  
FROM Viticulteur
```

# Requête simple : projection

## ■ Syntaxe de la clause **SELECT**

- Expressions et (re)nommage :  $\text{Exp}_1 [ \text{AS} \text{ Colonne}_1 ]$ ,  $\text{Exp}_2 [ \text{AS} \text{ Colonne}_2 ]$ , ...

- *Créer de nouvelles colonnes en combinant les valeurs des anciennes*

La spécification d'un nom pour identifier la colonne est souvent nécessaire

Viticulteur	<u>NUM</u>	Nom	Prénom	Région	CA
1	Barré	Henri	Bordeaux	100000	
3	Hermelin	Jean-Pierre	Champagne	150000	
4	Fort	Valérie	Languedoc	100000	
6	Schmidt	Thomas	Bourgogne	200000	

	<u>NUM</u>	<b>CA</b>
1		85000
3		127500
4		85000
6		170000

Prévoir pour chaque viticulteur un  
*nouveau chiffre d'affaire*  
considérant une récession de 15%

```
SELECT Num, CA * 85 / 100 AS CA  
FROM Viticulteur
```



# Requête simple : projection

## ■ Syntaxe de la clause **SELECT**

- Expressions et (re)nommage :  $\text{Exp}_1 [ \text{AS} \text{ Colonne}_1 ]$ ,  $\text{Exp}_2 [ \text{AS} \text{ Colonne}_2 ]$ , ...

- *Créer de nouvelles colonnes en combinant les valeurs des anciennes*

La spécification d'un nom pour identifier la colonne est souvent nécessaire

- *Renommer les attributs*

Viticulteur	<u>NUM</u>	Nom	Prénom	Région	CA
	1	Barré	Henri	Bordeaux	100000
	3	Hermelin	Jean-Pierre	Champagne	150000
	4	Fort	Valérie	Languedoc	100000
	6	Schmidt	Thomas	Bourgogne	200000

	NUM	CA	OldCA
	1	85000	100000
	3	127500	150000
	4	85000	100000
	6	170000	200000

Prévoir pour chaque viticulteur un nouveau chiffre d'affaire considérant une récession de 15%, et *conserver l'ancienne valeur pour comparaison*

```
SELECT Num, CA * 85 / 100 AS CA, CA AS OldCA
FROM Viticulteur
```

# Requête simple : projection

## ■ Syntaxe de la clause **SELECT**

- Expressions et (re)nommage :  $Exp_1 [ AS Colonne_1 ]$ ,  $Exp_2 [ AS Colonne_2 ]$ , ...

- *Créer de nouvelles colonnes en combinant les valeurs des anciennes*

La spécification d'un nom pour identifier la colonne est souvent nécessaire

- *Renommer les attributs*

Viticulteur	<u>NUM</u>	Nom	Prénom	Région	CA
	1	Barré	Henri	Bordeaux	100000
	3	Hermelin	Jean-Pierre	Champagne	150000
	4	Fort	Valérie	Languedoc	100000
	6	Schmidt	Thomas	Bourgogne	200000

	NUM	CA	OldCA	Devise
	1	85000	100000	euro
	3	127500	150000	euro
	4	85000	100000	euro
	6	170000	200000	euro

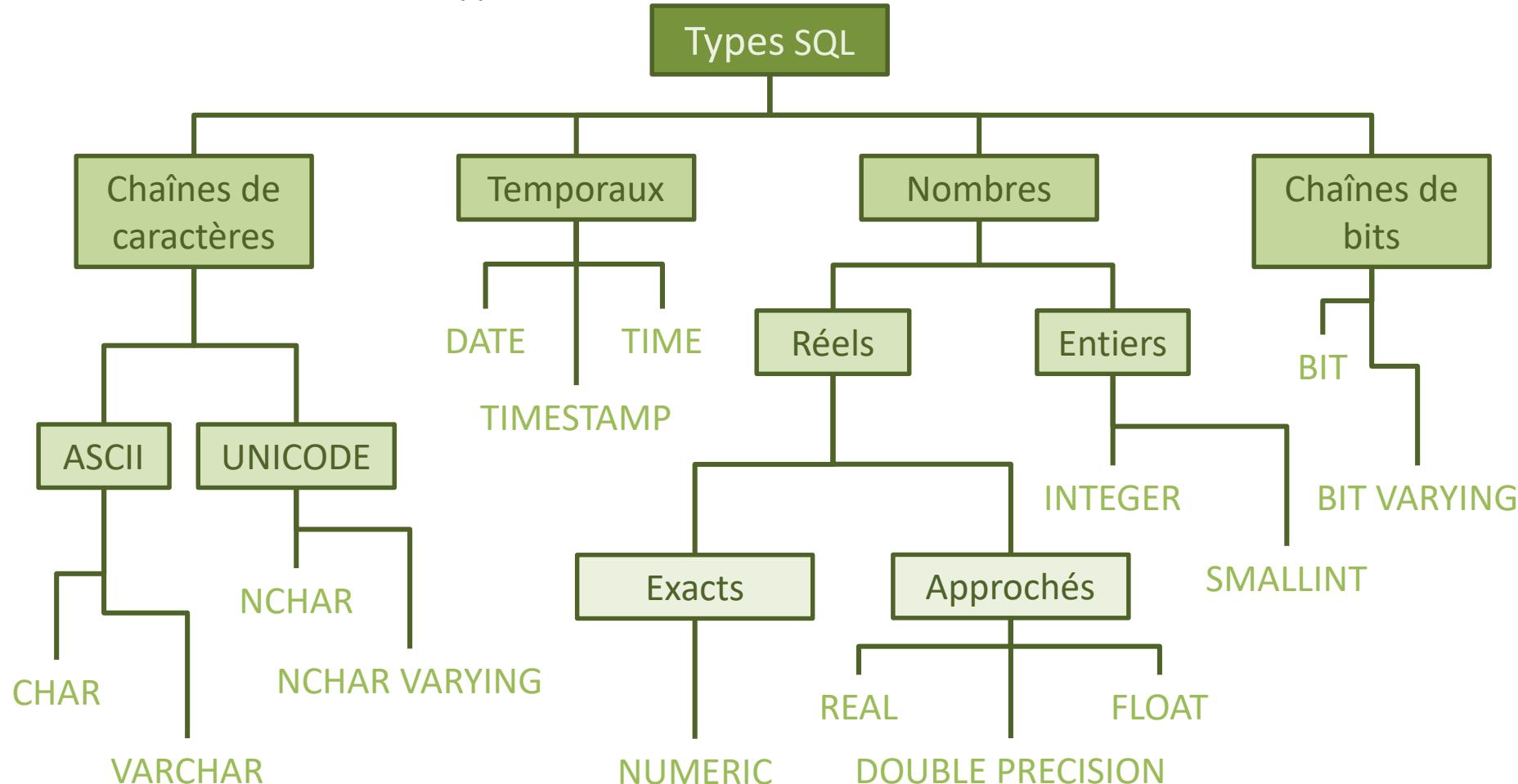
Prévoir pour chaque viticulteur un nouveau chiffre d'affaire considérant une récession de 15%, et conserver l'ancienne valeur pour comparaison ; *préciser que la devise est l'euro*

```
SELECT Num, CA * 85 / 100 AS CA ,  
      CA AS OldCA , 'euro' AS Devise  
FROM Viticulteur
```

# Requête simple : projection

## ■ Syntaxe des expressions Exp

### □ Hiérarchie des types SQL



# Requête simple : projection

## ■ Syntaxe des expressions Exp

### □ Types numériques

#### ■ Sous-typage

- **INT / INTEGER / SMALLINT** : entiers standards / standards / courts
- **REAL / DOUBLE PRECISION** : flottant standard simple / double précision
- **FLOAT(*n*)** : représentation binaire des réels avec précision *n*
- **NUMERIC(*p, d*)** : représentation décimale des réels
  - p* : nombre de chiffres, *d* : position de la virgule
  - NUMERIC(3,1)** représente **44.5** mais pas **444.5** ou **0.32**

#### ■ Constantes Cte<sub>Num</sub> : représentation classique avec le « . » pour virgule

#### ■ Quelques opérations

- Opérations arithmétiques : **Exp<sub>Num</sub> { + | - | \* | / } Exp<sub>Num</sub> | - Exp<sub>Num</sub>**
- Autres fonctions (absentes de SQL-92)
  - abs, log, sqrt, sign, round, power, mod, floor, ...**

# Requête simple : projection

## ■ Syntaxe des expressions Exp

- Chaînes de caractères
  - Sous-typage
    - CHAR(*n*) : chaîne ASCII de longueur fixe *n*
    - VARCHAR(*n*) : chaîne ASCII de longueur variable avec taille max. *n*
    - CHAR(*n*) : chaîne UNICODE (National) de longueur fixe *n*
    - ...
  - Constantes Cte<sub>Str</sub> : suite de caractères entre simple quote  
Chaîne vide ", 'Codd', 'L"expression' (apostrophe : deux quotes '')
  - Quelques opérations
    - Concaténation : Exp<sub>Str</sub> || Exp<sub>Str</sub>  
*Prenom* || '' || *Nom*
    - Mise en minuscule/majuscule : LOWER(Exp<sub>Str</sub>) | UPPER(Exp<sub>Str</sub>)  
*Prenom* || '' || UPPER(*Nom*)
    - Sous-chaîne : SUBSTRING(Exp<sub>Str</sub> FROM Exp<sub>Num</sub> FOR Exp<sub>Num</sub>)

# Requête simple : projection

## ■ Syntaxe des expressions Exp

- Types temporels
  - Sous-typepage
    - DATE : représentation des dates
    - TIME / TIME(*p*) : représentation des horaires (précision *p* pour les sec.)
    - TIMESTAMP / TIMESTAMP(*p*) : représentation des dates avec horaires
  - Constantes Cte<sub>Date</sub>
    - CURRENT\_DATE, CURRENT\_TIME, CURRENT\_TIMESTAMP
    - DATE 'YYYY-MM-DD', TIME 'HH:MM:SS.SSS', TIMESTAMP '...'  
TIMESTAMP '2014-01-27 09:11:45.6789' avec *p* = 4
  - Opération : extraction d'informations temporelles (Exp<sub>Num</sub>)  
`EXTRACT({ year | month | day | hour | minute | second } FROM ExpDate)`  
`EXTRACT(year FROM DATE '2014-01-27') => 2014`  
`EXTRACT(second FROM TIME '09:11:45.6789') => 45.6789`  
`EXTRACT(day FROM TIMESTAMP '2014-01-27 09:11:45.6789') => 27`

# Requête simple : projection

## ■ Définition de la grammaire

RequeteSimple ::= **SELECT** [ **ALL** | **DISTINCT** ] Projection **FROM** Source ...

Source ::= *Table* | ...

Projection ::= \* | Exp<sub>1</sub> [ **AS** *Colonne*<sub>1</sub> ], Exp<sub>2</sub> [ **AS** *Colonne*<sub>2</sub> ], ...

Exp ::= *Attribut* | (Exp)

| Cte<sub>Num</sub> | Exp<sub>Num</sub> { + | - | \* | / } Exp<sub>Num</sub> | - Exp<sub>Num</sub>

| Cte<sub>Str</sub> | Exp<sub>Str</sub> || Exp<sub>Str</sub> | { LOWER | UPPER }(Exp<sub>Str</sub>)

| SUBSTRING(Exp<sub>Str</sub> **FROM** Exp<sub>Num</sub> **FOR** Exp<sub>Num</sub>)

| CURRENT\_DATE | CURRENT\_TIME | DATE Exp<sub>Str</sub> | TIME Exp<sub>Str</sub> | ...

| EXTRACT({ year | month | day | hour | ... } **FROM** Exp<sub>Date</sub>)

| ...

# Requête simple : syntaxe générale

## ■ Grammaire des requêtes simples

Composition de *clauses*

- une projection (**SELECT**)
- une sélection (**WHERE**)
- une construction d'une table source (**FROM**)
- un groupement avec sélection (**GROUP BY ... HAVING ...**)
- un tri (**ORDER BY**)

```
RequeteSimple ::= SELECT Projection  
                  FROM Source  
                  [ WHERE Condition ]  
                  [ GROUP BY Groupement  
                    [ HAVING Condition ] ]  
                  [ ORDER BY Ordre ]
```

Source ::= *Table* | ...

# Requête simple : sélection

## ■ Syntaxe de la clause WHERE

- Un prédicat : Predicat
  - *Ne conserver que les lignes d'une table vérifiant une certaine propriété*
  - Différents prédicats
    - Comparaison : Exp { = | <> | < | > | <= | >= } Exp

Vin	NVin	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux	✓
2	Champagne	1996	3	Champagne	✗
3	Pauillac	1992	1	Bordeaux	✓
4	Chablis	2007	6	Bourgogne	✗

```
SELECT * FROM Vin  
WHERE Region = 'Bordeaux'
```

$\tilde{\sigma}_{Region} = \langle\!\langle Bordeaux \rangle\!\rangle (Vin)$

	NVin	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux	
3	Pauillac	1992	1	Bordeaux	

# Requête simple : sélection

## ■ Syntaxe de la clause WHERE

- Un prédicat : Predicat
  - *Ne conserver que les lignes d'une table vérifiant une certaine propriété*
  - Différents prédicats
    - Comparaison : Exp { = | <> | < | > | <= | >= } Exp

La différence est  
notée  $\neq$  et non  $\neq$

**SELECT \* FROM Vin**  
**WHERE Region  $\neq$  'Bordeaux'**

Vin	NVin	Cru	Millésime	Viticulteur	Région
1	Saint Emilion	2002	1	Bordeaux	<span style="color: red;">X</span>
2	Champagne	1996	3	Champagne	<span style="color: green;">✓</span>
3	Pauillac	1992	1	Bordeaux	<span style="color: red;">X</span>
4	Chablis	2007	6	Bourgogne	<span style="color: green;">✓</span>

$\tilde{\sigma}_{\text{Region} \neq \text{'Bordeaux}}(\text{Vin})$

	NVin	Cru	Millésime	Viticulteur	Région
2	Champagne	1996	3	Champagne	
4	Chablis	2007	6	Bourgogne	

# Requête simple : sélection

## ■ Syntaxe de la clause WHERE

- Un prédicat : Predicat
  - *Ne conserver que les lignes d'une table vérifiant une certaine propriété*
  - Différents prédicats
    - Comparaison : Exp { = | <> | < | > | <= | >= } Exp

L'ordre dépend du type  
Lexicographique pour les  
chaînes de caractères

```
SELECT * FROM Vin
WHERE Region > 'Cha'
```

Vin	NVin	Cru	Millésime	Viticulteur	Région
1	Saint Emilion	2002	1	Bordeaux	<span style="color:red">X</span>
2	Champagne	1996	3	Champagne	<span style="color:green">✓</span>
3	Pauillac	1992	1	Bordeaux	<span style="color:red">X</span>
4	Chablis	2007	6	Bourgogne	<span style="color:red">X</span>

$\tilde{\sigma}_{Region > \text{« Cha »} (Vin)}$

	NVin	Cru	Millésime	Viticulteur	Région
2	Champagne	1996	3	Champagne	

# Requête simple : sélection

## ■ Syntaxe de la clause WHERE

- Un prédicat : Predicat
  - *Ne conserver que les lignes d'une table vérifiant une certaine propriété*
  - Différents prédicats
    - Comparaison : Exp { = | <> | < | > | <= | >= } Exp

L'ordre dépend du type  
Lexicographique pour les  
chaînes de caractères

```
SELECT * FROM Vin
WHERE Region <= 'Bourgogne'
```

Vin	NVin	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux	✓
2	Champagne	1996	3	Champagne	✗
3	Pauillac	1992	1	Bordeaux	✓
4	Chablis	2007	6	Bourgogne	✓

$\tilde{\sigma}_{Region} \leq \text{« Bourgogne »} (Vin)$

	NVin	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux	
3	Pauillac	1992	1	Bordeaux	
4	Chablis	2007	6	Bourgogne	

# Requête simple : sélection

## ■ Syntaxe de la clause **WHERE**

- Un prédicat : **Predicat**
  - *Ne conserver que les lignes d'une table vérifiant une certaine propriété*
  - Différents prédicats
    - Inclusion dans un intervalle : **Exp [NOT] BETWEEN Exp AND Exp**

Vin	NVin	Cru	Millésime	Viticulteur	Région
-----	------	-----	-----------	-------------	--------

```
SELECT * FROM Vin  
WHERE Millesime  
BETWEEN 2000 AND 2007
```

1	Saint-Emilion	2002	1	Bordeaux	✓
2	Champagne	1996	3	Champagne	✗
3	Pauillac	1992	1	Bordeaux	✗
4	Chablis	2007	6	Bourgogne	✓

Les bornes sont incluses

NVin	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux
4	Chablis	2007	6	Bourgogne

# Requête simple : sélection

## ■ Syntaxe de la clause WHERE

- Un prédicat : Predicat
  - *Ne conserver que les lignes d'une table vérifiant une certaine propriété*
  - Différents prédicats
    - *Pattern matching : Exp<sub>Str</sub> [NOT] LIKE Exp<sub>Str</sub>*  
« % » n'importe quelle chaîne, « \_ » n'importe quel caractère

Vin	NVin	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux	✓
2	Champagne	1996	3	Champagne	✗
3	Pauillac	1992	1	Bordeaux	✗
4	Chablis	2007	6	Bourgogne	✓

```
SELECT * FROM Vin  
WHERE Cru NOT LIKE '__a%'
```

Les crus dont la troisième  
lettre n'est pas un « a »

	NVin	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux	
4	Chablis	2007	6	Bourgogne	

# Requête simple : sélection

## ■ Syntaxe de la clause **WHERE**

- Un prédictat : **Predicat**
  - *Ne conserver que les lignes d'une table vérifiant une certaine propriété*
  - Différents prédictats
    - Cas spécial de la valeur **NULL** : **Exp IS [NOT] NULL**  
attribut non-défini ; **NULL** ne peut pas être utilisée comme constante

```
SELECT * FROM Client  
WHERE Mail IS NOT NULL
```

Client	<u>NUM</u>	Nom	Prénom	Région	Mail
2	Voser	Armande	Alsace	⊥	
3	Hermelin	Jean-Pierre	Champagne	jph@champ.fr	
5	Senard	Danièle	Champagne	⊥	
6	Schmidt	Thomas	Bourgogne	tschmi@gmail.com	

	<u>NUM</u>	Nom	Prénom	Région	Mail
3	Hermelin	Jean-Pierre	Champagne	jph@champ.fr	
6	Schmidt	Thomas	Bourgogne	tschmi@gmail.com	

# Requête simple : sélection

## ■ Syntaxe de la clause WHERE

- Composition de prédicats : Condition

- *Ne conserver que les lignes d'une table vérifiant une formule complexe*

- Formules supposées en *forme normale disjonctive*

- Condition<sub>1</sub> OR Condition<sub>2</sub> OR ... | Condition<sub>1</sub> AND Condition<sub>2</sub> AND ...
    - | NOT Condition

Priorité : négation > conjonction > disjonction

```
SELECT * FROM Vin
WHERE
    NOT Viticulteur = 1
    AND NVin <> 2
    OR Millesime = 2002
```

Vin	NVin	Cru	Millésime	Viticulteur	Région
-----	------	-----	-----------	-------------	--------

1	Saint-Emilion	2002	1	Bordeaux
2	Champagne	1996	3	Champagne
3	Pauillac	1992	1	Bordeaux
4	Chablis	2007	6	Bourgogne

	NVin	Cru	Millésime	Viticulteur	Région
--	------	-----	-----------	-------------	--------

1	Saint-Emilion	2002	1	Bordeaux
4	Chablis	2007	6	Bourgogne

# Requête simple : sélection

## ■ Syntaxe de la clause **WHERE**

- Composition de prédicats : **Condition**

- *Ne conserver que les lignes d'une table vérifiant une formule complexe*

- Cas de base des formules

- { **Predicat** | **(Condition)** } [ **IS [NOT]** { **TRUE** | **FALSE** | **UNKNOWN** } ]

- NULL apparaît dans un calcul/prédicat, le résultat est **NULL/UNKNOWN**

Client	<u>NUM</u>	Nom	Prénom	Région	Mail
2	Voser	Armande	Alsace	⊥	
3	Hermelin	Jean-Pierre	Champagne	jph@champ.fr	
5	Senard	Danièle	Champagne	⊥	
6	Schmidt	Thomas	Bourgogne	tschmi@gmail.com	

```
SELECT * FROM Client  
WHERE  
Mail IS NOT NULL
```

```
SELECT * FROM Client  
WHERE  
Mail = NULL
```

```
SELECT * FROM Client  
WHERE  
Mail = NULL IS UNKNOWN
```

# Requête simple : sélection

## ■ Syntaxe de la clause **WHERE**

- Composition de prédictats : Condition

- *Ne conserver que les lignes d'une table vérifiant une formule complexe*

- Cas de base des formules

- { Predicat | (Condition) } [ IS [NOT] { TRUE | FALSE | UNKNOWN } ]

- Logique à trois valeurs

x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

# Requête simple : sélection

## ■ Définition de la grammaire

RequeteSimple ::= **SELECT** ... **FROM** Source **WHERE** Condition ...

Source ::= *Table* | ...

Condition ::= Condition<sub>1</sub> **OR** Condition<sub>2</sub> **OR** ... | Condition<sub>1</sub> **AND** Condition<sub>2</sub> **AND** ...  
| **NOT** Condition  
| { Predicat | (Condition) } [ **IS** [**NOT**] { TRUE | FALSE | UNKNOWN } ]

Predicat ::= Exp { = | <> | < | > | <= | >= } Exp  
| Exp [**NOT**] **BETWEEN** Exp **AND** Exp  
| Exp<sub>Str</sub> [**NOT**] **LIKE** Exp<sub>Str</sub>  
| Exp **IS** [**NOT**] **NULL**  
| ...

# Requête simple : syntaxe générale

## ■ Grammaire des requêtes simples

Composition de *clauses*

- une projection (**SELECT**)
- une sélection (**WHERE**)
- une construction d'une table source (**FROM**)
- un groupement avec sélection (**GROUP BY ... HAVING ...**)
- un tri (**ORDER BY**)

```
RequeteSimple ::= SELECT Projection  
                  FROM ListeSources  
                  [ WHERE Condition ]  
                  [ GROUP BY Groupement ]  
                  [ HAVING Condition ]  
                  [ ORDER BY Ordre ]
```

# Requête simple : produit cartésien, jointures

## ■ Syntaxe de la clause **FROM**

- Produits cartésiens :  $\text{Source}_1 \{ , | \text{CROSS JOIN } \} \text{Source}_2 \{ , | \text{CROSS JOIN } \} \dots$   
*Croiser l'ensemble des lignes de différentes sources*

```
SELECT *  
FROM Commande, Client
```

ou

```
SELECT *  
FROM Commande CROSS JOIN Client
```

	NCom	NClient	NVin	Date	Quantité	NUM	Nom	Prénom	Région
1	5	4	27/04/12	25	2	Voser	Armande		Alsace
1	5	4	27/04/12	25	3	Hermelin	Jean-Pierre		Champagne
1	5	4	27/04/12	25	5	Senard	Danièle		Champagne
1	5	4	27/04/12	25	6	Schmidt	Thomas		Bourgogne
2	3	2	25/01/13	100	2	Voser	Armande		Alsace
...	...	...	...	...	...	...	...	...	...

*Commande  $\bowtie$  Client*

# Requête simple : produit cartésien, jointures

## ■ Syntaxe de la clause **FROM**

- Jointures :  $\text{Source}_1 \text{ JointureOp } \text{Source}_2 \text{ JointureOp } \dots [\text{ON Condition}]$   
 $\text{JointureOp} ::= [\text{NATURAL}] [\text{INNER} | \{\text{RIGHT} | \text{LEFT} | \text{FULL}\} \text{OUTER}] \text{JOIN}$

LES TYPES DE JOINTURES		Naturelle <b>NATURAL</b>	$\theta$ -jointure <b>ON</b>
Interne <b>INNER</b>		$\bowtie$ ... <b>NATURAL INNER JOIN</b> ...	$\bowtie_\theta$ ... <b>INNER JOIN</b> ... <b>ON</b> ...
Externe <b>OUTER</b>	à gauche <b>LEFT</b>	$\overset{\circ}{\bowtie}_L$ ... <b>NATURAL LEFT OUTER JOIN</b> ...	$\overset{\circ}{\bowtie}_\theta L$ ... <b>LEFT OUTER JOIN</b> ... <b>ON</b> ...
	à droite <b>RIGHT</b>	$\overset{\circ}{\bowtie}_R$ ... <b>NATURAL RIGHT OUTER JOIN</b> ...	$\overset{\circ}{\bowtie}_\theta R$ ... <b>RIGHT OUTER JOIN</b> ... <b>ON</b> ...
	bilatérale <b>FULL</b>	$\overset{\circ}{\bowtie}$ ... <b>NATURAL FULL OUTER JOIN</b> ...	$\overset{\circ}{\bowtie}_\theta$ ... <b>FULL OUTER JOIN</b> ... <b>ON</b> ...

# Requête simple : produit cartésien, jointures

## ■ Syntaxe de la clause **FROM**

- Jointures :  $\text{Source}_1 \text{ JointureOp } \text{Source}_2 \text{ JointureOp } \dots [\text{ON Condition}]$   
 $\text{JointureOp ::= [ NATURAL ] [ INNER | { RIGHT | LEFT | FULL } OUTER ] JOIN}$

```
SELECT *
FROM Vin, Commande
WHERE Commande.NVin = Vin.NVin
AND Quantite > 30
```

≡

```
SELECT *
FROM Vin INNER JOIN Commande
ON Commande.NVin = Vin.NVin
AND Quantite > 30
```

$\tilde{\sigma}_{\text{Commande.NVin}=\text{Vin.NVin} \wedge \text{Quantité}>30} (\text{Vin} \bowtie \text{Commande})$

$\text{Vin} \bowtie_{\text{Commande.NVin}=\text{Vin.NVin} \wedge \text{Quantité}>30} \text{Commande}$

	Vin. Nvin	Cru	Millésime	Viticu- liteur	Région	NCom	NClient	Commande. NVin	Date	Quantité
1	Saint-Emilion	2002	1	Bordeaux	4	5	1	08/11/10	100	
2	Champagne	1996	3	Champagne	2	3	2	25/01/13	100	
2	Champagne	1996	3	Champagne	3	2	2	14/08/09	80	

# Requête simple : produit cartésien, jointures

## ■ Syntaxe de la clause **FROM**

- Jointures :  $\text{Source}_1 \text{ JointureOp } \text{Source}_2 \text{ JointureOp } \dots [\text{ON Condition}]$   
 $\text{JointureOp ::= [ NATURAL ] [ INNER | { RIGHT | LEFT | FULL } OUTER ] JOIN}$

```
SELECT *
FROM Vin NATURAL INNER JOIN Commande
WHERE Quantite > 30
```

$$\tilde{\sigma}_{\text{Quantité} > 30}(\text{Vin} \bowtie \text{Commande})$$

	Cru	Millésime	Viticu-lteur	Région	Nvin	NCom	NClient	Date	Quantité
Saint-Emilion	2002	1	Bordeaux	1	4	5	08/11/10	100	
Champagne	1996	3	Champagne	2	2	3	25/01/13	100	
Champagne	1996	3	Champagne	2	3	2	14/08/09	80	

# Requête simple : produit cartésien, jointures

## ■ Syntaxe de la clause **FROM**

- Variable de tuple : { *Table* | (*ListeSources*) | ... } [ [AS] *tupleVar* ]
  - *Chaque tuple désigné par les sources peut être nommé*  
Le nom par défaut est celui de la table
  - Objectif 1 : éviter *les noms à rallonge...*

```
SELECT * FROM Vin, Commande  
WHERE Commande.NVin = Vin.NVin AND Quantite > 30
```

≡

```
SELECT * FROM Vin v, Commande c  
WHERE c.NVin = v.NVin AND Quantite > 30
```

# Requête simple : produit cartésien, jointures

## ■ Syntaxe de la clause **FROM**

- Variable de tuple : { *Table* | (*ListeSources*) | ... } [ [AS] *tupleVar* ]
  - *Chaque tuple désigné par les sources peut être nommé*  
Le nom par défaut est celui de la table
  - Objectif 2 : *lever les ambiguïtés*  
*Trouver les régions produisant au moins deux vins*

```
SELECT Vin.Region FROM Vin, Vin
WHERE Vin.Region = Vin.Region AND Vin.NVin <> Vin.NVin
```

```
SELECT v1.Region FROM Vin v1, Vin v2
WHERE v1.Region = v2.Region AND v1.NVin <> v2.NVin
```

# Requête simple : produit cartésien, jointures

## ■ Définition de la grammaire

RequeteSimple ::= **SELECT** ... **FROM** ListeSources ...

ListeSources ::= Source<sub>1</sub> { , | **CROSS JOIN** } Source<sub>2</sub> { , | **CROSS JOIN** } ...  
| Source<sub>1</sub> JointureOp Source<sub>2</sub> JointureOp ... [**ON** Condition]

JointureOp ::= [ **NATURAL** ] [ **INNER** | { **RIGHT** | **LEFT** | **FULL** } **OUTER** ] **JOIN**

Source ::= { *Table* | (ListeSources) | ... } [ [AS] *tupleVar* ]

# Requête simple : syntaxe générale

## ■ Grammaire des requêtes simples

Composition de *clauses*

- une projection (**SELECT**)
- une sélection (**WHERE**)
- une construction d'une table source (**FROM**)
- un groupement avec sélection (**GROUP BY ... HAVING ...**)
- un tri (**ORDER BY**)

```
RequeteSimple ::= SELECT Projection  
                  FROM ListeSources  
                  [ WHERE Condition ]  
                  [ GROUP BY Groupement  
                    [ HAVING Condition ] ]  
                  [ ORDER BY Ordre ]
```

# Requête simple : groupement

## ■ Syntaxe de la clause **GROUP BY ... HAVING ...**

- Suite d'attributs : *Attribut<sub>1</sub>, Attribut<sub>2</sub>, ...*

*Les tuples partageant les mêmes valeurs pour ces attributs sont regroupés*

Pas d'affichage du groupement possible ...

**SELECT \* FROM *Commande* GROUP BY *NVin***

$\gamma_{NVin}(\textit{Commande})$

Commande	<u>NCom</u>	NClient	<u>NVin</u>	Date	Quantité		<u>NCom</u>	NClient	<u>NVin</u>	Date	Quantité
1	5	4	27/04/12	25		1	5	4	27/04/12	25	
2	3	2	25/01/13	100		2	3	2	25/01/13	100	
3	2	2	14/08/09	80		3	2	2	14/08/09	80	
4	5	1	08/11/10	100		5	3	1	16/07/13	30	
5	3	2	16/07/13	30		4	5	1	08/11/10	100	

# Requête simple : groupement

## ■ Syntaxe de la clause **GROUP BY ... HAVING ...**

- Suite d'attributs : *Attribut<sub>1</sub>, Attribut<sub>2</sub>, ...*

*Les tuples partageant les mêmes valeurs pour ces attributs sont regroupés*

Pas d'affichage du groupement possible ...

```
SELECT * FROM Commande GROUP BY NVin, NClient
```

$\gamma_{NVin, NClient}(\text{Commande})$

Commande	NCom	NClient	NVin	Date	Quantité		NCom	NClient	NVin	Date	Quantité
1	5	4	27/04/12	25		1	5	4	27/04/12	25	
2	3	2	25/01/13	100		2	3	2	25/01/13	100	
3	2	2	14/08/09	80		5			16/07/13	30	
4	5	1	08/11/10	100		3	2	2	14/08/09	80	
5	3	2	16/07/13	30		4	5	1	08/11/10	100	

# Requête simple : groupement

## ■ Syntaxe de la clause **GROUP BY ... HAVING ...**

- Agrégations : { AVG | MAX | MIN | SUM | COUNT } ([ ALL | DISTINCT ]Exp)

*Expressions permettant d'agréger les valeurs au sein d'un même groupe*

Projeter sur les attributs groupés et les attributs non groupés mais agrégés

```
SELECT NVin, AVG(Quantite) AS QtMoy FROM Commande GROUP BY NVin
```

$\gamma_{NVin, AVG(Quantité)} \rightarrow QtMoy$  (*Commande*)

	NCom	NClient	NVin	Date	Quantité
1	5	4	27/04/12	25	
2	3		25/01/13	100	
3	2	2	14/08/09	80	
5	3		16/07/13	30	
4	5	1	08/11/10	100	

	NVin	QtMoy
4	25	
2	70	
1	100	

# Requête simple : groupement

## ■ Syntaxe de la clause **GROUP BY ... HAVING ...**

- Agrégations : { AVG | MAX | MIN | SUM | COUNT } ([ ALL | DISTINCT ]Exp)

*Expressions permettant d'agréger les valeurs au sein d'un même groupe*

Projeter sur les attributs groupés et les attributs non groupés mais agrégés

```
SELECT NVin, COUNT(NClient) AS NbClient FROM Commande GROUP BY NVin
```

$\gamma_{NVin, CNT(NClient) \rightarrow NbClient} (Commande)$

	NCom	NClient	NVin	Date	Quantité
1	5	4	27/04/12	25	
2	3		25/01/13	100	
3	2	2	14/08/09	80	
5	3		16/07/13	30	
4	5	1	08/11/10	100	

	NVin	NbClient
4	1	
2	3	
1	1	

# Requête simple : groupement

## ■ Syntaxe de la clause **GROUP BY ... HAVING ...**

- Agrégations : { AVG | MAX | MIN | SUM | COUNT } ([ ALL | DISTINCT ]Exp)

*Expressions permettant d'agréger les valeurs au sein d'un même groupe*

Projeter sur les attributs groupés et les attributs non groupés mais agrégés

```
SELECT NVin, COUNT(DISTINCT NClient) AS NbClient  
FROM Commande  
GROUP BY NVin
```

	NCom	NClient	NVin	Date	Quantité
1	5	4	27/04/12	25	
2	3		25/01/13	100	
3	2	2	14/08/09	80	
5	3		16/07/13	30	
4	5	1	08/11/10	100	

	NVin	NbClient
4	1	
2	2	
1	1	

# Requête simple : groupement

## ■ Syntaxe de la clause **GROUP BY ... HAVING ...**

- Groupement conditionné : **HAVING Condition**

*Ne conserver que les groupes vérifiant une certaine propriété*

```
SELECT NVin, COUNT(DISTINCT NCom) AS NbCom  
FROM Commande  
GROUP BY Nvin  
HAVING AVG(Quantite) < 80
```

	NCom	NClient	NVin	Date	Quantité
	1	5	4	27/04/12	25
	2	3		25/01/13	100
	3	2	2	14/08/09	80
	5	3		16/07/13	30
	4	5	1	08/11/10	100

	NVin	NbCom
	4	1
	2	3
	1	1

✓	4	1
✓	2	3
✗	1	1

	NVin	NbCom
	4	1
	2	3

# Requête simple : groupement

## ■ Syntaxe de la clause **GROUP BY ... HAVING ...**

- Groupement conditionné : **HAVING Condition**

*Ne conserver que les groupes vérifiant une certaine propriété*

```
SELECT NVin, COUNT(DISTINCT NCom) AS NbCom  
FROM Commande  
WHERE Quantite > 30  
GROUP BY Nvin
```

	NCom	NClient	NVin	Date	Quantité
	1	5	4	27/04/12	25
	2	3		25/01/13	100
	3	2	2	14/08/09	80
	5	3		16/07/13	30
	4	5	1	08/11/10	100

✗ ✓ ✓ ✗ ✗

	NVin	NbCom
	2	2
	1	1

# Requête simple : groupement

## ■ Syntaxe de la clause **GROUP BY ... HAVING ...**

- Groupement conditionné : **HAVING Condition**

*Ne conserver que les groupes vérifiant une certaine propriété*

```
SELECT NVin, COUNT(DISTINCT NCom) AS NbCom  
FROM Commande  
WHERE Quantite > 30  
GROUP BY Nvin  
HAVING AVG(Quantite) < 80
```

	NCom	NClient	NVin	Date	Quantité
	1	5	4	27/04/12	25
	2	3		25/01/13	100
	3	2	2	14/08/09	80
	5	3		16/07/13	30
	4	5	1	08/11/10	100

✗

✓

✓

✗

✓

NVin	NbCom
2	2
1	1

✗

✗

NVin	NbCom

# Requête simple : groupement

## ■ Syntaxe de la clause **GROUP BY ... HAVING ...**

### □ Exercice

*Crus et millésimes des vins commandés au plus 1 fois avec leurs quantités*

# Requête simple : groupement

## ■ Syntaxe de la clause **GROUP BY ... HAVING ...**

### □ Exercice

*Crus et millésimes des vins commandés au plus 1 fois avec leurs quantités*

```
SELECT Cru, Millesime, COUNT(Quantite) AS Quantite  
FROM Commande c NATURAL RIGHT OUTER JOIN Vin v  
GROUP BY Nvin  
HAVING COUNT(DISTINCT NCom) <= 1
```

# Requête simple : groupement

## ■ Définition de la grammaire

RequeteSimple ::= **SELECT** ... **GROUP BY** Groupement **HAVING** Condition ...

Groupement ::= *Attribut<sub>1</sub>*, *Attribut<sub>2</sub>*, ...

Exp ::= ...  
| { AVG | MAX | MIN | SUM | COUNT } ([ ALL | DISTINCT ] Exp)  
| ...

# Requête simple : syntaxe générale

## ■ Grammaire des requêtes simples

Composition de *clauses*

- une projection (**SELECT**)
- une sélection (**WHERE**)
- une construction d'une table source (**FROM**)
- un groupement avec sélection (**GROUP BY ... HAVING ...**)
- un tri (**ORDER BY**)

```
RequeteSimple ::= SELECT Projection  
                  FROM ListeSources  
                  [ WHERE Condition ]  
                  [ GROUP BY Groupement ]  
                  [ HAVING Condition ]  
                  [ ORDER BY Ordre ]
```

# Requête simple : tri

## ■ Syntaxe de la clause **ORDER BY**

- Ordre lexicographique : *Colonne<sub>1</sub> [ASC|DESC], Colonne<sub>2</sub> [ASC|DESC], ...*

*Après projection, les lignes sont triées suivant un ordre spécifié*

*Lister les couples clients (« NOM Prénom ») / fournisseurs (« NOM Prénom ») ayant déjà commercé trié par clients croissants et fournisseurs décroissants*

**SELECT DISTINCT**

*(UPPER(cl.Nom) || ' ' || cl.Prenom) AS Client,  
(UPPER(vi.Nom) || ' ' || vi.Prenom) AS Fournisseur*

**FROM** *Client cl INNER JOIN Commande c INNER JOIN Vin v INNER JOIN Viticulteur vi  
ON cl.NUM = c.NClient AND c.NVin = v.NVin AND v.Viticulteur = vi.NUM*

**ORDER BY** *Client ASC, Fournisseur DESC*

# Requête simple : tri

## ■ Définition de la grammaire

RequeteSimple ::= **SELECT** ... **ORDER BY** Ordre

Ordre ::= *Colonne<sub>1</sub>* [**ASC|DESC**], *Colonne<sub>2</sub>* [**ASC|DESC**], ...

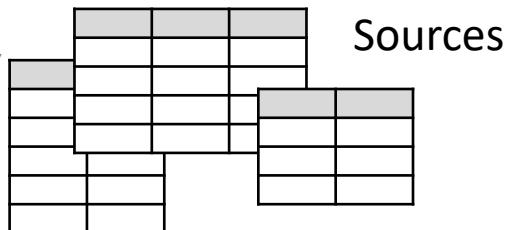
# Requête simple

## ■ Sémantique



```
SELECT Exp1 AS Col1, SUM(Exp2) AS Col2, ...  
FROM Source1 some JOIN ...  
WHERE Condition1  
GROUP BY Attr1, ...  
HAVING Condition2  
ORDER BY Col1 ASC, ...
```

⋈, ⋈°, ×, ...



# Requête simple

## Sémantique



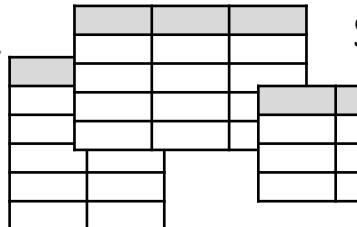
$\sigma_{\text{Condition}_1}$



✓ ✓ ✓ ✗ ✓ ✓ ✗ ✓

```
SELECT Exp1 AS Col1, SUM(Exp2) AS Col2, ...
FROM Source1 some JOIN ...
WHERE Condition1
GROUP BY Attr1, ...
HAVING Condition2
ORDER BY Col1 ASC, ...
```

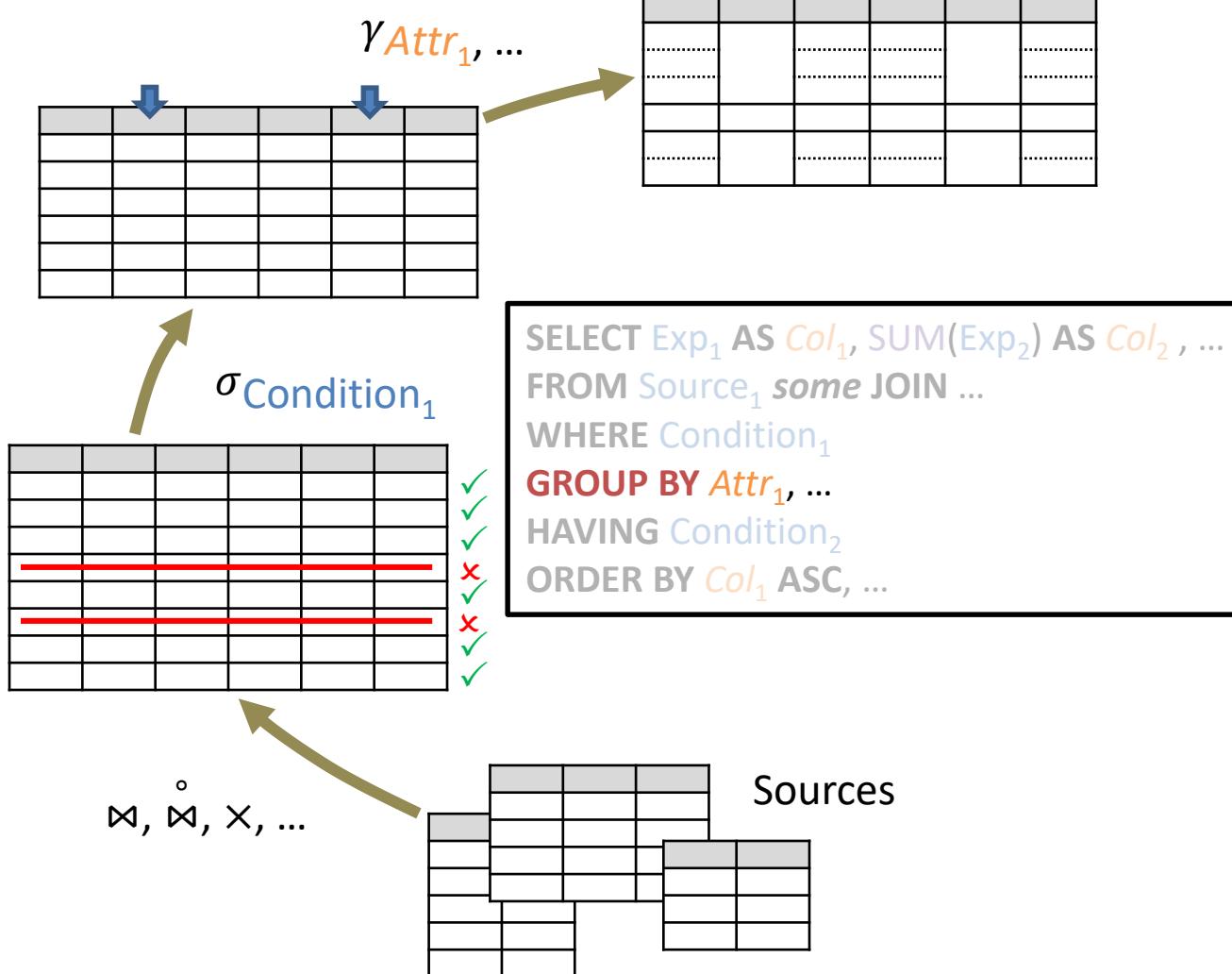
$\bowtie, \bowtie^\circ, \times, \dots$



Sources

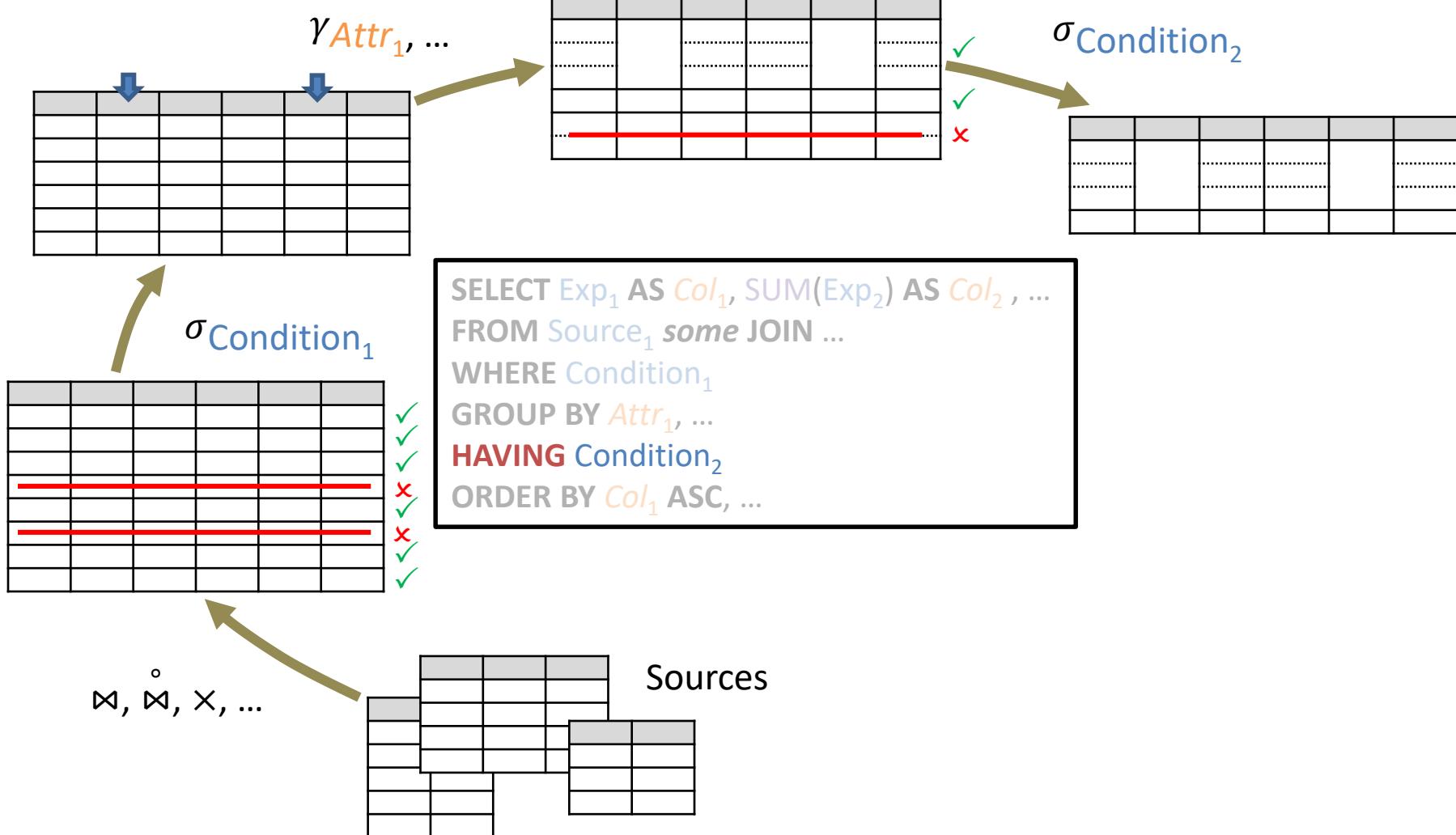
# Requête simple

# Sémantique



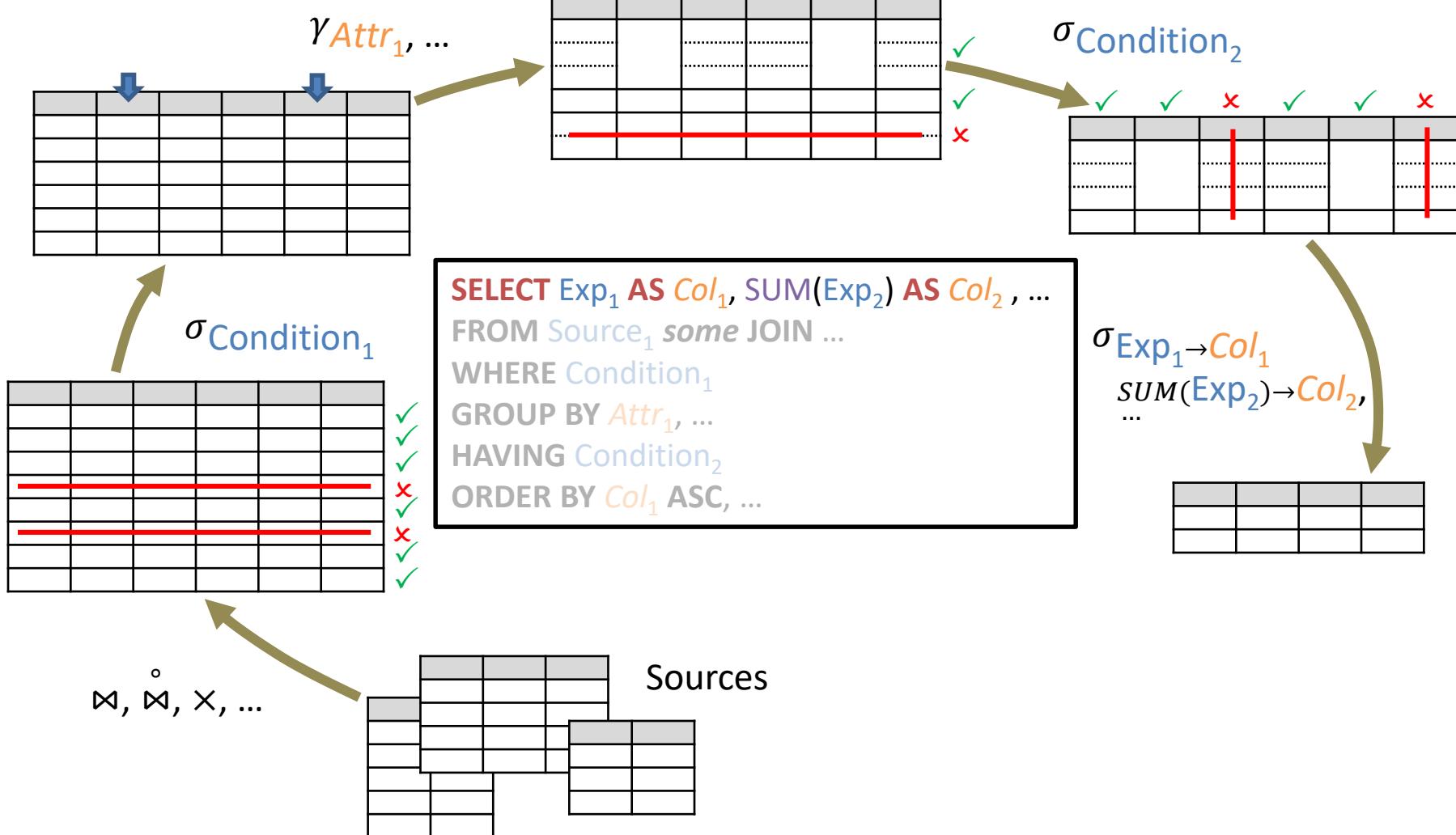
# Requête simple

## Sémantique



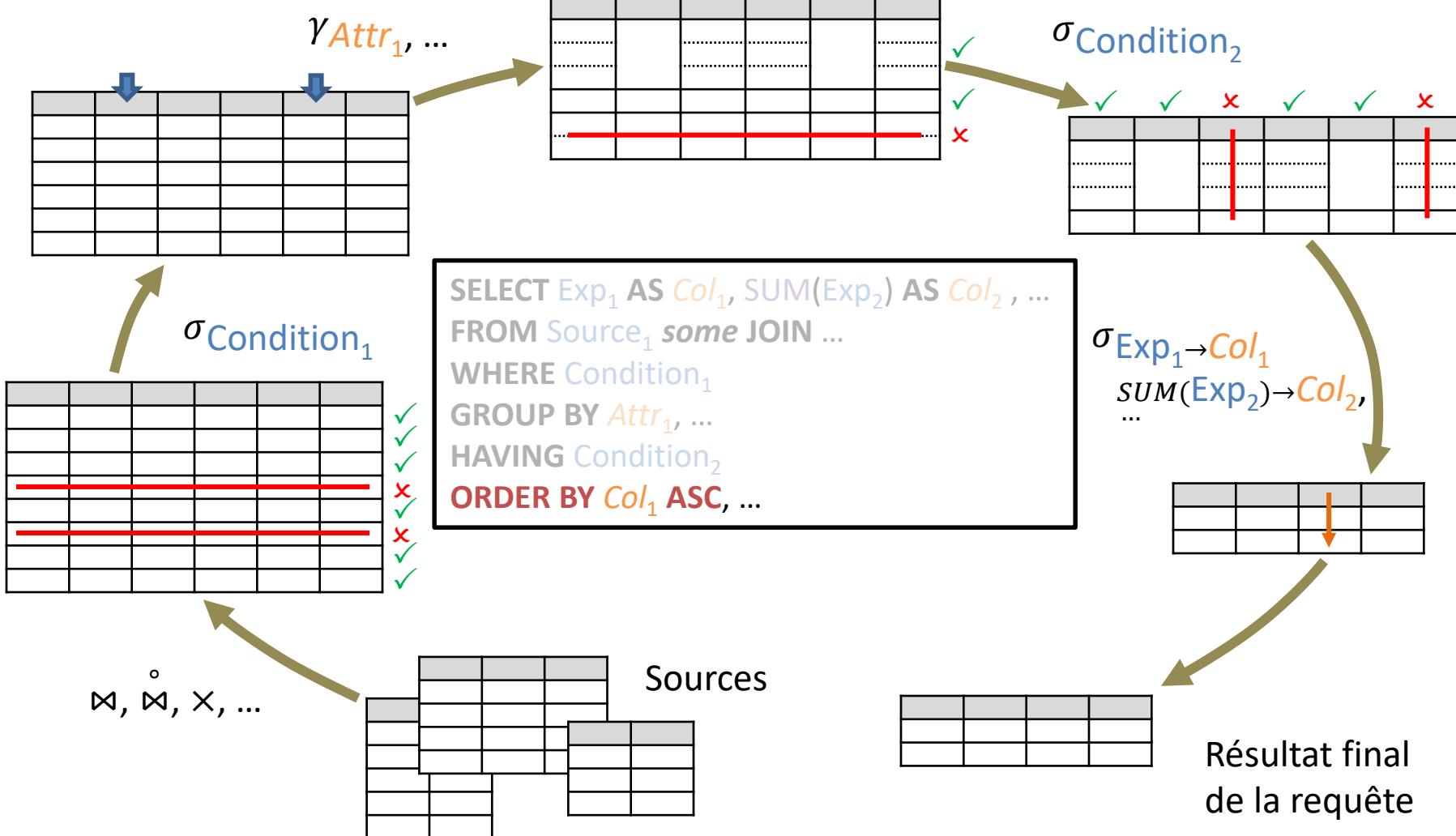
# Requête simple

## Sémantique



# Requête simple

## Sémantique



# Plan



- Requêtes simples
- Compositions de requêtes
- Petit guide pratique

# Composition de requêtes : op. booléennes

## ■ Syntaxe des opérateurs UNION, INTERSECT et EXCEPT

- Attention, chaque opérande doit posséder le même schéma

Viticulteur	<u>NUM</u>	Nom	Prénom	Région	CA
1	Barré	Henri		Bordeaux	100000
3	Hermelin	Jean-Pierre		Champagne	150000
4	Fort	Valérie		Languedoc	100000
6	Schmidt	Thomas		Bourgogne	200000

Client	<u>NUM</u>	Nom	Prénom	Région	Mail
2	Voser	Armande		Alsace	✉
3	Hermelin	Jean-Pierre		Champagne	jph@champ.fr
5	Senard	Danièle		Champagne	✉
6	Schmidt	Thomas		Bourgogne	tschmi@gmail.com

# Composition de requêtes : op. booléennes

## ■ Syntaxe des opérateurs UNION, INTERSECT et EXCEPT

### □ Union de requêtes

*Faire l'union des tuples issus de deux requêtes*

```
(SELECT Nom, Prenom, Region FROM Client)  
UNION  
(SELECT Nom, Prenom, Region FROM Viticulteur)
```

*Client U Viticulteur*

	Nom	Prénom	Région
Barré	Henri	Bordeaux	
Voser	Armande	Alsace	
Hermelin	Jean-Pierre	Champagne	
Fort	Valérie	Languedoc	
Senard	Danièle	Champagne	
Schmidt	Thomas	Bourgogne	

Lister l'ensemble des personnes (nom, prénom, région) référencées dans la base

# Composition de requêtes : op. booléennes

## ■ Syntaxe des opérateurs UNION, INTERSECT et EXCEPT

- Union *disjointe* de requêtes

*Faire l'union des tuples issus de deux requêtes sans enlever les doublons*

```
(SELECT Nom, Prenom, Region FROM Client)  
UNION ALL  
(SELECT Nom, Prenom, Region FROM Viticulteur)
```

*Client  $\cup$  Viticulteur*

	Nom	Prénom	Région
Barré	Henri	Bordeaux	
Voser	Armande	Alsace	
Hermelin	Jean-Pierre	Champagne	
Hermelin	Jean-Pierre	Champagne	
Fort	Valérie	Languedoc	
Senard	Danièle	Champagne	
Schmidt	Thomas	Bourgogne	
Schmidt	Thomas	Bourgogne	

Lister l'ensemble des personnes (nom, prénom, région) référencées dans la base

# Composition de requêtes : op. booléennes

## ■ Syntaxe des opérateurs UNION, INTERSECT et EXCEPT

### □ Intersection de requêtes

*Faire l'intersection des tuples issus de deux requêtes*

```
(SELECT Nom, Prenom, Region FROM Client)  
INTERSECT  
(SELECT Nom, Prenom, Region FROM Viticulteur)
```

*Client ∩ Viticulteur*

	Nom	Prénom	Région
Hermelin	Jean-Pierre	Champagne	
Schmidt	Thomas	Bourgogne	

Lister les personnes à la fois clientes et viticultrices

# Composition de requêtes : op. booléennes

## ■ Syntaxe des opérateurs UNION, INTERSECT et EXCEPT

- Différence de requêtes

*Faire la différence entre les tuples issus de deux requêtes*

```
(SELECT Nom, Prenom, Region FROM Client)  
EXCEPT  
(SELECT Nom, Prenom, Region FROM Viticuliteur)
```

Client \ Viticulleur

	Nom	Prénom	Région
Voser	Armande	Alsace	
Senard	Danièle	Champagne	

Lister les personnes clientes mais pas viticultrices

# Composition de requêtes : op. booléennes

## ■ Grammaire des opérations booléennes

```
Requete ::= RequeteSimple  
          | (Requete1) BoolOp (Requete2) BoolOp ...
```

```
BoolOp ::= { UNION | INTERSECT | EXCEPT } [ ALL ]
```

# Composition de requêtes : sous-requêtes

## ■ Retours possibles d'une requête

- Soit vide : aucun tuple n'a été trouvé
- Soit NULL : absence d'information, erreur, ...
- Soit une seule donnée : une seule ligne, une seule colonne
- Soit une table avec une seule ligne
- Soit une table avec plusieurs lignes

# Composition de requêtes : sous-requêtes

## ■ En tant que valeur

- Définition d'une colonne :  $\text{Exp} ::= \dots | (\text{Requete})$

*Calculer la valeur d'une colonne en fonction du résultat d'une requête*

```
SELECT Nom, Prenom, (SELECT COUNT(DISTINCT NCom)
                      FROM Commande
                      WHERE NClient = NUM) AS NbCom
       FROM Client
```

Trouver pour chaque client  
(nom, prénom) son nombre de  
commandes passées

	Nom	Prénom	NbCom
Voser	Armande	1	
Hermelin	Jean-Pierre	2	
Senard	Danièle	2	
Schmidt	Thomas	0	

# Composition de requêtes : sous-requêtes

## ■ En tant que valeur

- Définition d'une colonne : `Exp ::= ... | (Requete)`

*Calculer la valeur d'une colonne en fonction du résultat d'une requête*

```
SELECT Nom, Prenom, COUNT(DISTINCT NCom ) AS NbCom  
FROM Client LEFT OUTER JOIN Commande  
ON NClient = NUM  
GROUP BY NClient
```

Trouver pour chaque client  
(nom, prénom) son nombre de  
commandes passées

	Nom	Prénom	NbCom
Voser	Armande	1	
Hermelin	Jean-Pierre	2	
Senard	Danièle	2	
Schmidt	Thomas	0	

# Composition de requêtes : sous-requêtes

## ■ En tant que valeur

- Utilisation dans les conditions : `Exp ::= ... | (Requete)`

*Utiliser le résultat d'une requête dans une condition*

```
SELECT Nom, Prenom, CA FROM Viticulteur
WHERE CA >= (SELECT CA FROM Viticulteur
WHERE Nom = 'Hermelin')
```

Trouver les viticulteurs  
dont le chiffre d'affaire est  
supérieur à celui d'Hermelin

	Nom	Prénom	CA
	Hermelin	Jean-Pierre	150000
	Schmidt	Thomas	200000

Viticulteur	NUM	Nom	Prénom	Région	CA
-------------	-----	-----	--------	--------	----

1	Barré	Henri	Bordeaux	100000
3	Hermelin	Jean-Pierre	Champagne	150000
4	Fort	Valérie	Languedoc	100000
6	Schmidt	Thomas	Bourgogne	200000

# Composition de requêtes : sous-requêtes

## ■ En tant que valeur

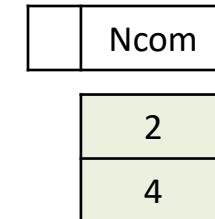
- Utilisation dans les conditions : `Exp ::= ... | (Requete)`

*Utiliser le résultat d'une requête dans une condition*

Trouver les numéros de commande  
dont les quantités sont les plus grandes

```
SELECT DISTINCT NCom FROM Commande  
WHERE Quantite >= (SELECT MAX(Quantite)  
FROM Commande)
```

Commande	<u>NCom</u>	NClient	<u>NVin</u>	Date	Quantité
1	5	4	27/04/12	25	
2	3	2	25/01/13	100	
3	2	2	14/08/09	80	
4	5	1	08/11/10	100	
5	3	2	16/07/13	30	



# Composition de requêtes : sous-requêtes

## ■ En tant que table

- Utilisation comme source : **Source ::= { ... | (Requete) } [ [AS] tupleVar ]**

*Utiliser le résultat d'une requête dans la clause **FROM** d'une autre requête*

```
SELECT Nom, Prenom, Cru
FROM
  ((SELECT Nom, Prenom, Region FROM Client) UNION
   (SELECT Nom, Prenom, Region FROM Viticulteur)) AS p
RIGHT OUTER JOIN
  (SELECT Cru, Region FROM Vin) AS v
ON p.Region <> v.Region
```

Lister pour chaque personne les crus des régions différentes de la leur

# Composition de requêtes : sous-requêtes

## ■ En tant que table

- Prédicat d'existence : **EXISTS(Requete)**

*Vrai si la requête retourne au moins un tuple*

```
SELECT Nom, Prenom  
FROM Client  
WHERE EXISTS(SELECT Nclient FROM Commande  
         WHERE Nclient = NUM)
```

Liste des clients ayant déjà passé commande

	Nom	Prénom
Voser	Armande	
Hermelin	Jean-Pierre	
Senard	Danièle	

# Composition de requêtes : sous-requêtes

## ■ En tant que table

- Prédicat d'appartenance : **Exp [NOT] IN (Requete)**

*Vrai si la valeur apparaît au moins une fois dans le résultat de la requête*

```
SELECT Nom, Prenom  
FROM Client  
WHERE NUM IN (SELECT Nclient FROM Commande)
```

Liste des clients ayant déjà passé commande

	Nom	Prénom
Voser	Armande	
Hermelin	Jean-Pierre	
Senard	Danièle	

# Composition de requêtes : sous-requêtes

## ■ En tant que table

- Prédicat d'appartenance : **Exp [NOT] IN (Requete)**

*Vrai si la valeur apparaît au moins une fois dans le résultat de la requête*

```
SELECT Nom, Prenom  
FROM Client  
WHERE NUM NOT IN (SELECT Nclient FROM Commande)
```

	Nom	Prénom
Schmidt	Thomas	

Liste des clients n'ayant jamais passé commande

# Composition de requêtes : sous-requêtes

## ■ En tant que table

- Prédicat de comparaison existentielle :  $\text{Exp} \{ = | < > | \dots \}$  ANY (Requete)

*Vrai s'il existe un tuple du résultat de la requête vérifiant la comparaison*

Commande	<u>NCom</u>	NClient	<u>NVin</u>	Date	Quantité
1	5	4	27/04/12	25	
2	3	2	25/01/13	100	
3	2	2	14/08/09	80	
4	5	1	08/11/10	100	
5	3	2	16/07/13	30	

Trouver les vins ayant été commandé au moins une fois en quantité supérieure à 50

Vin	<u>Nvin</u>	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux	
2	Champagne	1996	3	Champagne	
3	Pauillac	1992	1	Bordeaux	
4	Chablis	2007	6	Bourgogne	

# Composition de requêtes : sous-requêtes

## ■ En tant que table

- Prédicat de comparaison existentielle :  $\text{Exp} \{ = | < > | \dots \}$  ANY (Requete)

*Vrai s'il existe un tuple du résultat de la requête vérifiant la comparaison*

Trouver les vins ayant été commandé au moins une fois en quantité supérieure à 50

```
SELECT * FROM Vin v
WHERE 50 <= ANY (SELECT Quantite
                  FROM Commande c)
                  WHERE c.NVin = v.NVin)
```

	Nvin	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux	
2	Champagne	1996	3	Champagne	

# Composition de requêtes : sous-requêtes

## ■ En tant que table

- Prédicat de comparaison universelle :  $\text{Exp} \{ = | < > | \dots \} \text{ALL}$  (Requete)

*Vrai si tous les tuples du résultat de la requête vérifient la comparaison*

Commande	<u>NCom</u>	<u>NClient</u>	<u>NVin</u>	Date	Quantité
1	5	4	27/04/12	25	
2	3	2	25/01/13	100	
3	2	2	14/08/09	80	
4	5	1	08/11/10	100	
5	3	2	16/07/13	30	

Trouver les vins ayant été  
Commandé à chaque fois  
en quantité supérieure à 50

Vin	<u>Nvin</u>	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux	
2	Champagne	1996	3	Champagne	
3	Pauillac	1992	1	Bordeaux	
4	Chablis	2007	6	Bourgogne	

# Composition de requêtes : sous-requêtes

## ■ En tant que table

- Prédicat de comparaison universelle :  $\text{Exp} \{ = | < > | \dots \} \text{ALL}$  (Requete)

*Vrai si tous les tuples du résultat de la requête vérifient la comparaison*

Trouver les vins ayant été  
Commandé à chaque fois  
en quantité supérieure à 50

```
SELECT * FROM Vin v
WHERE 50 <= ALL (SELECT Quantite
                  FROM Commande c
                  WHERE c.NVin = v.NVin)
```

	Nvin	Cru	Millésime	Viticulteur	Région
1	Saint-Emilion	2002	1	Bordeaux	
3	Pauillac	1992	1	Bordeaux	

# Composition de requêtes : sous-requêtes

## ■ Grammaire des sous-requêtes

Source ::= { ... | (Requete) } [ [AS] *tupleVar* ]

Exp ::= ... | (Requete)

Predicat ::= ...

- | EXISTS(Requete)
- | Exp [NOT] IN (Requete)
- | Exp { = | <> | ... } ANY (Requete)
- | Exp { = | <> | ... } ALL (Requete)

# Plan



- Requêtes simples
- Compositions de requêtes
- Petit guide pratique



-- FIN --