Ressource R1.05
Introduction aux bases de données et SQL

Lydia Boudjeloud-Assala
Professeure des Universités en Informatique

lydia.boudjeloud-assala@univ-lorraine.fr

Département Informatique - IUT Metz

2024-2025

Plan

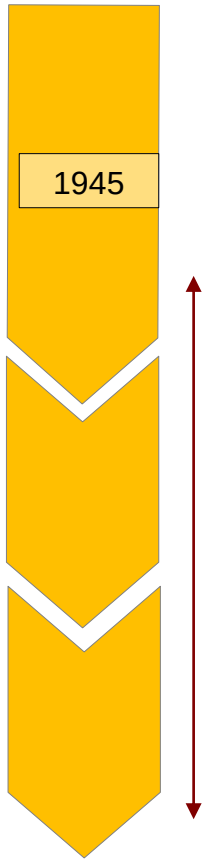
- Algèbre relationnelle
- Introduction au langage SQL

Langage et algèbre relationnel

Un peu d'histoire ….

fin de la 2nd guerre mondiale

?



Travaux de Turing sur l'informatique (machine universelle)

A. Turing

essor de l'informatique

Augmentation de puissance,
Augmentation des usages
(Informatique d'entreprise)

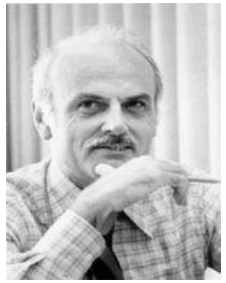
Comment **stocker** et **retrouver efficacement** les données dans un ordinateur ?

Un peu d'histoire ….

Émergence du
modèle relationnel

1970

Publication.
"A Relational Model of Data for Large Shared Data Banks",
E.F. Codd



E.F Codd

1981

E.F. Codd reçoit le prix Turing
pour ses travaux sur le modèle relationnel

Norme **SQL** 1
(minimal)

1986

***S**tructured **Q**uery **L**anguage*
(Langage de requête structuré)

Introduction

Le langage SQL (*Structured Query Language*)

- Il a fait l'objet de plusieurs normes ANSI/ISO dont la plus répandue aujourd'hui est la norme SQL2 qui a été définie en 1992.

Année	Nom	Appellation	Commentaires
1986	ISO/CEI 9075:1986	SQL-86 ou SQL-87	Édité par l'ANSI puis adopté par l'ISO en 1987.
1989	ISO/CEI 9075:1989	SQL-89 ou SQL-1	Révision mineure.
1992	ISO/CEI 9075:1992	SQL-92 (en) alias SQL2	Révision majeure.
1999	ISO/CEI 9075:1999	SQL-99 (en) alias SQL3	Expressions rationnelles, requêtes récursives, déclencheurs, types non-scalaires et quelques fonctions orientées objet (les deux derniers points sont quelque peu controversés et pas encore largement implémentés).
2003	ISO/CEI 9075:2003	SQL:2003 (en)	Introduction de fonctions pour la manipulation XML, « window functions », ordres standardisés et colonnes avec valeurs auto-produites (y compris colonnes d'identité).
2008	ISO/CEI 9075:2008	SQL:2008 (en)	Ajout de quelques fonctions de fenêtrage (ntile, lead, lag, first value, last value, nth value), limitation du nombre de lignes (OFFSET / FETCH), amélioration mineure sur les types distincts, curseurs et mécanismes d'auto-incrémentation.
2011	ISO/CEI 9075:2011	SQL:2011 (en)	Ajout du support des tables temporelles (historisation automatique).

https://fr.wikipedia.org/wiki/Structured_Query_Language

Introduction

Le langage SQL (*Structured Query Language*)

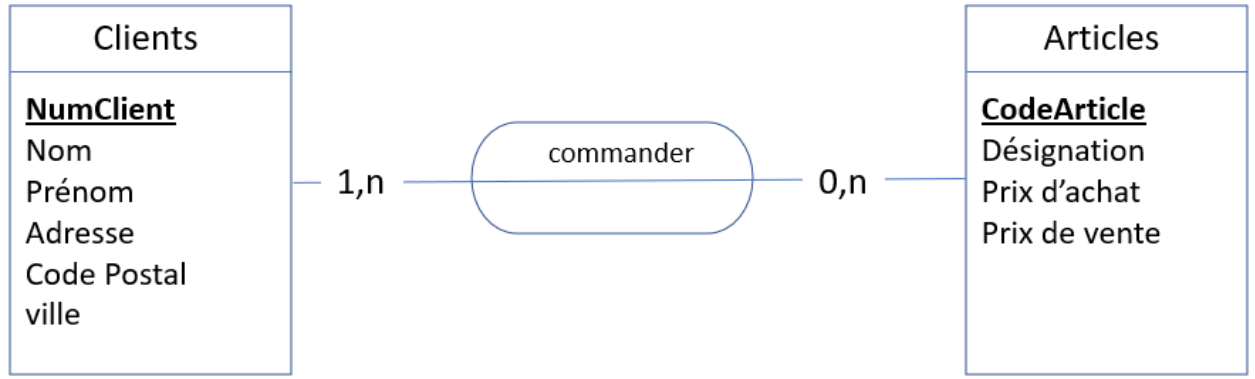
- Il a fait l'objet de plusieurs normes ANSI/ISO dont la plus répandue aujourd'hui est la norme SQL2 qui a été définie en 1992.

Année	Nom	Appellation	Commentaires
2011	ISO/CEI 9075:2011	SQL:2011 (en)	Ajout du support des tables temporelles (historisation automatique).
2016	ISO/IEC 9075:2016	SQL:2016	Ajout la correspondance des motifs de ligne et les fonctions de table polymorphes, ainsi que le support JSON tant attendu. La norme SQL a ajouté la prise en charge de JSON pour permettre l'interopérabilité avec les applications modernes et les nouveaux types de bases de données.
2019	ISO/IEC 9075:2019	SQL:2019	Elle a ajouté la partie qui définit la prise en charge des tableaux multidimensionnels dans SQL.
2023	ISO/IEC 9075:2023	SQL:2023	Une série de modifications du langage SQL existant, de nouvelles fonctionnalités pour JSON et l'introduction d'une nouvelle section pour les requêtes sur les graphes de propriétés pour définir et parcourir les graphes dans un SGBD relationnel.

Quelques rappels

MCD – MLD – Schéma Relationnel

MCD



MLD - MRD

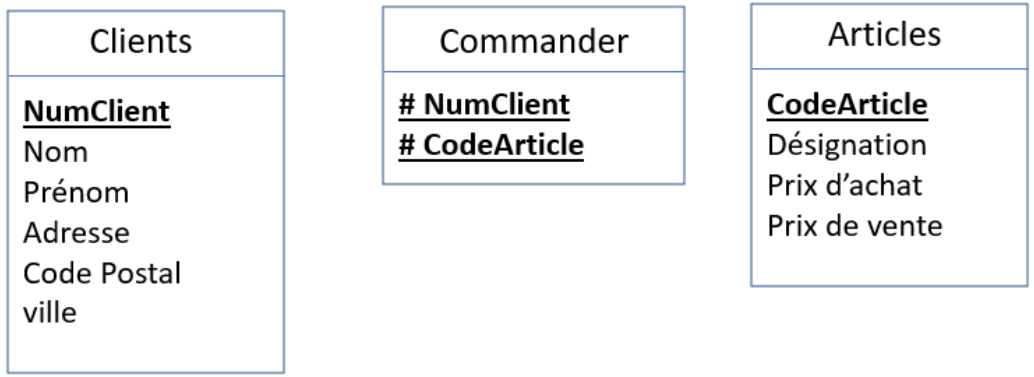


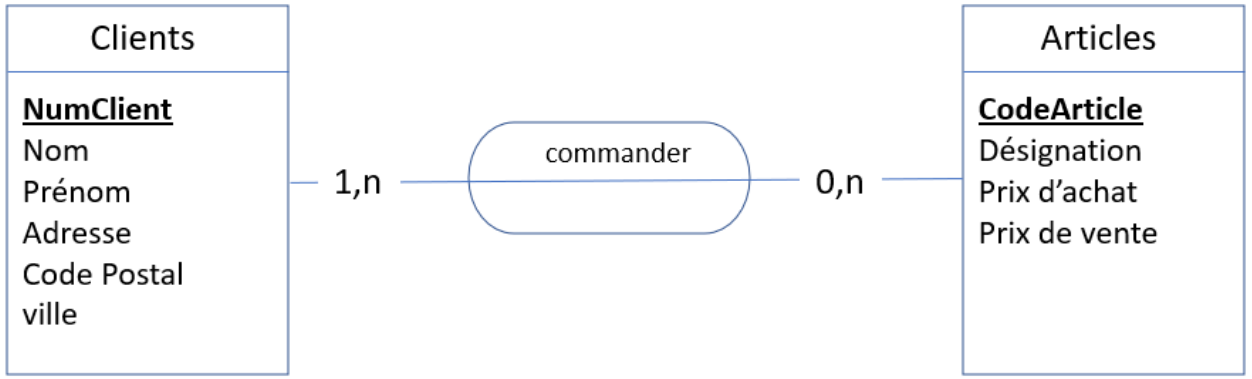
Schéma relationnel

Clients (**NumClient,** Nom, Prénom, Adresse, Code Postal, ville)
Articles (**CodeArticle,** Désignation, Prix d'achat, Prix de vente)
Commander (**# NumClient, # CodeArticle**)

MCD – MLD – Schéma Relationnel

MCD

E/A



MLD - MRD

Schéma relationnel Relations

Clients (**NumClient,** Nom, Prénom, Adresse, Code Postal, ville)
Articles (**CodeArticle,** Désignation, Prix d'achat, Prix de vente)
Commander (**# NumClient, # CodeArticle**)

Tables
Relations

Quelques rappels

relation

Synonyme : table, tableau

nom	prénom	dateNaissance	sexe	taille	poids
Ah	Alice	1970-01-01	F	160	50
Beh	Bob	1970-01-03	M	175	70
Ceh	Charles	1980-02-05	M	183	85

Personne

nom	texte / caractères
prénom	texte / caractères
dateNaissance	date
sexe	texte / énumération...
taille	numérique
poids	numérique

Personne (nom , prénom , dateNaissance , Sexe , taille , poids)

Quelques rappels

Personne	
nom	texte / caractères ← Domaine de l'attribut
prénom	texte / caractères
dateNaissance	date
sexe	texte / énumération...
taille	numérique
poids	numérique

Personne(nom , prénom , dateNaissance , Sexe , taille , poids)

attribut de la relation

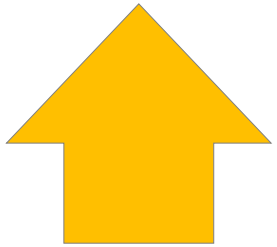


schéma (description) **de la relation**, indépendant des données quelle contient :

nom de la relation, suivi de la liste de **ses attributs** entre parenthèse, éventuellement complété de **leur domaine**.

A quoi ça sert

Exemple

```
geographie (
  numCommune, libelleCommune, chefLieuDepartement,
  numDepartement, libelleDepartement,
  numRegion, libelleRegion, population
);
```

num Commune	libelle Commune	chefLieu Departement	num Departement	libelle Departement	num Region	libelle Region	population
57001	Metz	oui	57	Moselle	44	Grand Est	
57672	Thionville	non	57	Moselle	44	Grand Est	

Exemple

```
geographie (
  numCommune, libelleCommune, chefLieuDepartement,
  numDepartement, libelleDepartement,
  numRegion, libelleRegion, population
);
```

num Commune	libelle Commune	chefLieu Departement	num Departement	libelle Departement	num Region	libelle Region	population
57001	Metz	oui	57	Moselle	***76***	***Occitanie***	
57672	Thionville	non	57	Moselle	44	Grand Est	

Anomalie *:*
1 département (Moselle) dans 2 régions différentes (Occitanie et Grand Est).

Exemple

```
geographie (
  numCommune, libelleCommune, chefLieuDepartement,
  numDepartement, libelleDepartement,
  numRegion, libelleRegion, population
);
```

num Commune	libelle Commune	chefLieu Departement	num Departement	libelle Departement	num Region	libelle Region	population
57001	Metz	oui	***58***	***Moselle***	44	Grand Est	
57672	Thionville	non	57	Moselle	44	Grand Est	

Anomalie *:*
1 département (Moselle) avec 2 numéros différents (57, 58).

Exemple

```
geographie (
  numCommune, libelleCommune, chefLieuDepartement,
  numDepartement, libelleDepartement,
  numRegion, libelleRegion, population
);
```

num Commune	libelle Commune	chefLieu Departement	num Departement	libelle Departement	num Region	libelle Region	population
57001	Metz	oui	57	Moselle	44	Grand Est	
57672	Thionville	***oui***	57	Moselle	44	Grand Est	

Anomalie *:*
2 chef lieux de département (Metz, Thionville)
pour 1 département (Moselle).

Exemple

num Commune	libelle Commune	chefLieu Departement	num Departement	libelle Departement	num Region	libelle Region	population
57001	Metz	oui	57	Moselle	44	Grand Est	
57672	Thionville	non	57	Moselle	44	Grand Est	

Ces anomalies viennent notamment de l'existence de dépendances fonctionnelles transitives

`numCommune → libelleCommune, chefLieuDepartement, numDepartement, libelleDepartement, numRegion, libelleRegion, population`

`numCommune → libelleCommune, chefLieuDepartement, numDepartement, population`

`numDepartement → libelleDepartement, numRegion`

`numRegion → libelleRegion`

Exemple

~~numCommune → libelleCommune, chefLieuDepartement, numDepartement, libelleDepartement, numRegion, libelleRegion, population~~

numCommune → libelleCommune, chefLieuDepartement, numDepartement, population

numDepartement → libelleDepartement, numRegion

numRegion → libelleRegion

Exemple

~~numCommune → libelleCommune, chefLieuDepartement, numDepartement, libelleDepartement, numRegion, libelleRegion, population~~

numCommune → libelleCommune, chefLieuDepartement, numDepartement, population

numDepartement → libelleDepartement, numRegion

numRegion → libelleRegion

communes(numCommune, libelleCommune, chefLieuDepartement, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, #numRegion) ;

regions(numRegion, libelleRegion) ;

Exemple

~~numCommune → libelleCommune, chefLieuDepartement, numDepartement, libelleDepartement, numRegion, libelleRegion, population~~

numCommune → libelleCommune, chefLieuDepartement, numDepartement, population

numDepartement → libelleDepartement, numRegion

numRegion → libelleRegion

communes(numCommune, libelleCommune, chefLieuDepartement, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, #numRegion) ;

regions(numRegion, libelleRegion) ;

~~1 table~~

Schéma d'une base de données = ensemble des schémas des relations
→ Ensemble de tables

Exemple

communes(numCommune, libelleCommune, chefLieuDepartement, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, #numRegion) ;

regions(numRegion, libelleRegion) ;

num Commune	libelle Commune	chefLieu Departement	num Departement	population
57001	Metz	oui	57	
57672	Thionville	non	57	

communes

num Departement	libelle Departement	num Region
56	Morbihan	53
57	Moselle	44
58	Nievre	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

Exemple

communes(numCommune, libelleCommune, chefLieuDepartement, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, #numRegion) ;

regions(numRegion, libelleRegion) ;

num Commune	libelle Commune	chefLieu Departement	num Departement	population
57001	Metz	oui	57	
57672	Thionville	non	57	

communes

num Departement	libelle Departement	num Region
56	Morbihan	53
57	Moselle	44
58	Nievre	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

Anomalies impossibles *:*
1 département (Moselle) dans 2 régions différentes (Occitanie et Grand Est).
1 département (Moselle) avec 2 numéros différents (57, 58).

Exemple

communes(numCommune, libelleCommune, chefLieuDepartement, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, #numRegion) ;

regions(numRegion, libelleRegion) ;

num Commune	libelle Commune	chefLieu Departement	num Departement	population
57001	Metz	oui	57	
57672	Thionville	non	57	

communes

num Departement	libelle Departement	num Region
56	Morbihan	53
57	Moselle	44
58	Nievre	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

il reste une anomalie possible *:*
1 département (Moselle) avec 2 chefs lieux.

Exemple

communes(numCommune, libelleCommune, chefLieuDepartement, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, #numRegion) ;

regions(numRegion, libelleRegion) ;

num Commune	libelle Commune	chefLieu Departement	num Departement	population
57001	Metz	oui	57	
57672	Thionville	non	57	

communes

num Departement	libelle Departement	num Region
56	Morbihan	53
57	Moselle	44
58	Nievre	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

il reste une anomalie possible *:*
1 département (Moselle) avec 2 chefs lieux.
Le chef lieu de département est une information sur le département : 1 département = 1 chef lieu

Exemple

communes(numCommune, libelleCommune, ~~chefLieuDepartement~~, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, numCommuneChefLieu, #numRegion) ;

regions(numRegion, libelleRegion) ;

departements[numcommuneChefLieu] ⊆ communes[numCommune]

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	

communes

num Departement	libelle Departement	numCommune ChefLieu	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

Anomalies impossibles *:*

- *1 département (Moselle) dans 2 régions différentes (Occitanie et Grand Est).*
- *1 département (Moselle) avec 2 numéros différents (57, 58).*
- *1 département avec 2 chefs lieux*

Exemple

communes(numCommune, libelleCommune, ~~chefLieuDepartement~~, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, numCommuneChefLieu, #numRegion) ;

regions(numRegion, libelleRegion) ;

departements[numcommuneChefLieu] ⊆ communes[numCommune]

Contrainte d'Intégrité Référentielle

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	

communes

num Departement	libelle Departement	numCommune ChefLieu	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

Anomalies impossibles *:*

- *1 département (Moselle) dans 2 régions différentes (Occitanie et Grand Est).*
- *1 département (Moselle) avec 2 numéros différents (57, 58).*
- *1 département avec 2 chefs lieux*

Conséquences

Stocker les données dans plusieurs relations permet

- une meilleure qualité des données (cohérence, règles métiers)
- un accès plus rapide aux données

Conséquences

Stocker les données dans plusieurs relations permet

- une meilleure qualité des données (cohérence, règles métiers)
- un accès plus rapide aux données

…

Mais

comment manipuler toutes ces données

Introduction

Introduction

Algèbre relationnelle :

- L'algèbre relationnelle est le support mathématique cohérent sur lequel repose le modèle relationnel
- L'algèbre relationnelle propose un ensemble d'opérations élémentaires formelles sur les relations dans le but de créer de nouvelles relations
- Ces opérations permettent de représenter des requêtes sur la base de données dont le résultat s'exprime sous la forme d'une relation (une table)

SQL :

- L'algèbre relationnelle est le formalisme qui est au cœur du langage de requête de SQL

Introduction

Arbre algébrique :

- **Requête :** Une question exprimée sous forme d'un programme d'opérations de l'algèbre relationnelle
- La requête peut être représentée par un arbre relationnel, ou arbre algébrique
  - Les nœuds correspondent aux représentations graphiques des opérations
  - Les arcs correspondent aux données liées par les opérations

Exemple pris de « Bases de donnée, Georges Gardarin, Eyrolles, 2003 »

Introduction

Arbre algébrique :

- **Requête :** Une question exprimée sous forme d'un programme d'opérations de l'algèbre relationnelle
- La requête peut être représentée par un arbre relationnel, ou arbre algébrique
  - Les nœuds correspondent aux représentations graphiques des opérations
  - Les arcs correspondent aux données liées par les opérations

Exemple pris de « Bases de donnée, Georges Gardarin, Eyrolles, 2003 »

Introduction

Arbre algébrique :

- **Requête :** Une **question exprimée** sous forme d'un programme d'opérations de l'algèbre relationnelle
- La requête peut être représentée par un arbre relationnel, ou arbre algébrique
  - Les nœuds correspondent aux représentations graphiques des opérations
  - Les arcs correspondent aux données liées par les opérations

Requête :
Noms et Prénoms des buveurs habitant Paris ayant bu du Chablis
depuis le 1er janvier 1992

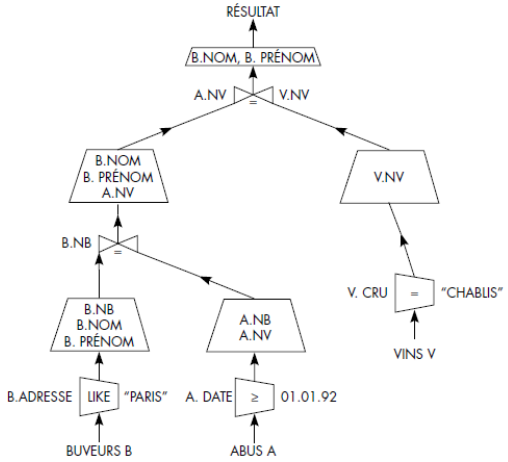
Exemple pris de « Bases de donnée, Georges Gardarin, Eyrolles, 2003 »

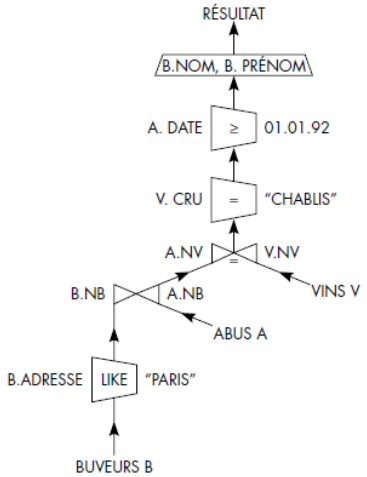
Introduction

Arbre algébrique :

- **Requête :** Une question exprimée sous forme d'un programme d'opérations de l'algèbre relationnelle
- La requête peut être représentée par un **arbre relationnel**, ou **arbre algébrique**
  - Les nœuds correspondent aux représentations graphiques des opérations
  - Les arcs correspondent aux données liées par les opérations

Requête :
Noms et Prénoms des buveurs habitant Paris ayant bu du Chablis depuis le 1er janvier 1992





Exemple pris de « Bases de donnée, Georges Gardarin, Eyrolles, 2003 »

Introduction

Arbre algébrique :

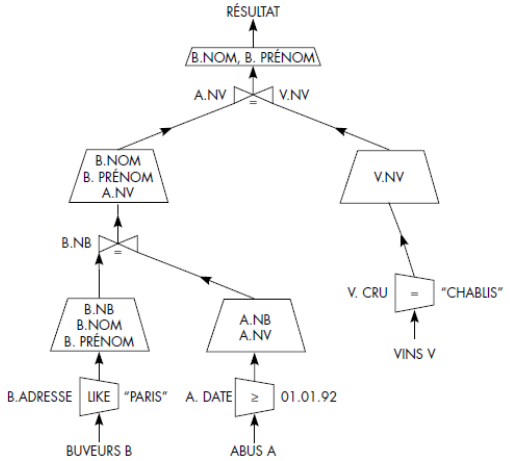
- **Requête :** Une question exprimée sous forme d'un programme d'opérations de l'algèbre relationnelle
- La requête peut être représentée par un arbre relationnel, ou arbre algébrique
  - Les nœuds correspondent aux représentations graphiques des opérations
  - Les arcs correspondent aux données liées par les opérations

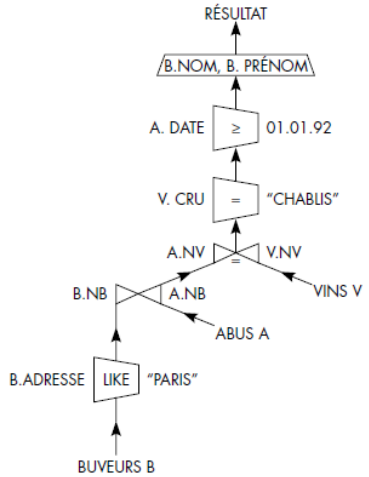
Requête :
Noms et Prénoms des buveurs habitant Paris ayant bu du Chablis depuis le 1er janvier 1992

À partir des 3 tables (Relations) : BUVEURS, ABUS, VINS

Schéma Relationnel :

BUVEURS (**NB**, NOM, PRENOM, ADRESSE, TYPE)
VINS (**NV**, CRU, MILL QUALITE, DEGRE)
ABUS (**#NB, #NV, DATE**, QUANTITE)

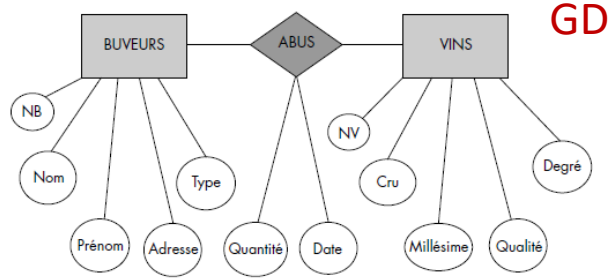




Exemple pris de « Bases de donnée, Georges Gardarin, Eyrolles, 2003 »

GDF

Introduction



Arbre algébrique :

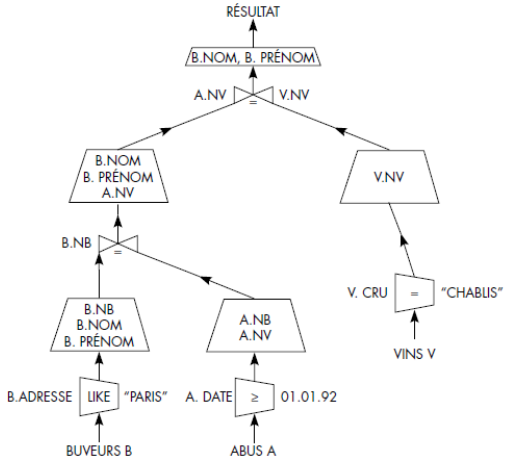
- **Requête :** Une question exprimée sous forme d'un programme d'opérations de l'algèbre relationnelle
- La requête peut être représentée par un arbre relationnel, ou arbre algébrique
  - Les nœuds correspondent aux représentations graphiques des opérations
  - Les arcs correspondent aux données liées par les opérations

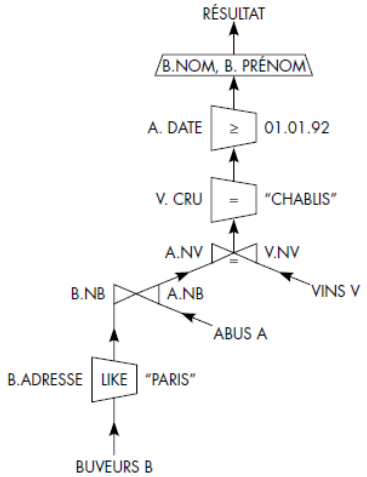
Requête :
Noms et Prénoms des buveurs habitant Paris ayant bu du Chablis depuis le 1er janvier 1992

À partir des 3 tables (Relations) : BUVEURS, ABUS, VINS

Schéma Relationnel :

BUVEURS (**NB**, NOM, PRENOM, ADRESSE, TYPE)
VINS (**NV**, CRU, MILL QUALITE, DEGRE)
ABUS (**#NB, #NV, DATE**, QUANTITE)





Exemple pris de « Bases de donnée, Georges Gardarin, Eyrolles, 2003 »

Introduction

On peut distinguer trois familles d'opérateurs relationnels :

Les opérateurs unaires (Sélection, Projection) :

Ce sont les opérateurs les plus simples, ils permettent de produire une nouvelle table à partir d'une autre table.

Les opérateurs binaires ensemblistes (Union, Intersection, Différence) :

Ces opérateurs permettent de produire une nouvelle relation à partir de deux relations de même degré (nombre d'attributs) et de même domaine (l'ensemble des valeurs possibles des attributs).

Les opérateurs binaires ou n-aires (Produit cartésien, Jointure, Division) :

Ils permettent de produire une nouvelle table à partir de deux ou plusieurs autres tables.

Les notations ne sont pas standardisées en algèbre relationnelle.
Ce cours utilise des notations courantes, mais donc pas forcément universelles.
L'algèbre relationnelle va nous amener petit à petit au langage SQL.

Introduction

On peut distinguer trois familles d'opérateurs relationnels :

Les opérateurs unaires (Sélection, Projection) :

Ce sont les opérateurs les plus simples, ils permettent de produire une nouvelle table à partir d'une autre table.

Les opérateurs binaires ensemblistes (Union, Intersection, Différence) :

Ces opérateurs permettent de produire une nouvelle relation à partir de deux relations de même degré (nombre d'attributs) et de même domaine (l'ensemble des valeurs possibles des attributs).

Les opérateurs binaires ou n-aires (Produit cartésien, Jointure, Division) :

Ils permettent de produire une nouvelle table à partir de deux ou plusieurs autres tables.

Les notations ne sont pas standardisées en algèbre relationnelle.
Ce cours utilise des notations courantes, mais donc pas forcément universelles.
L'algèbre relationnelle va nous amener petit à petit au langage SQL.

Introduction

On peut distinguer trois familles d'opérateurs relationnels :

Les opérateurs unaires (Sélection, Projection) :

Ce sont les opérateurs les plus simples, ils permettent de produire une nouvelle table à partir d'une autre table.

Les opérateurs binaires ensemblistes (Union, Intersection, Différence) :

Ces opérateurs permettent de produire une nouvelle relation à partir de deux relations de même degré (nombre d'attributs) et de même domaine (l'ensemble des valeurs possibles des attributs).

Au S2

Les opérateurs binaires ou n-aires (Produit cartésien, Jointure, Division) :

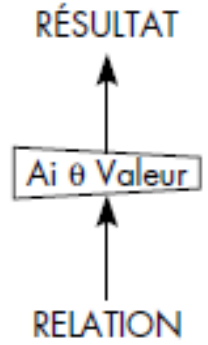
Ils permettent de produire une nouvelle table à partir de deux ou plusieurs autres tables.

Les notations ne sont pas standardisées en algèbre relationnelle.
Ce cours utilise des notations courantes, mais donc pas forcément universelles.
L'algèbre relationnelle va nous amener petit à petit au langage SQL (en parallèle).

Algèbre Relationnelle

Exemples de :
« Bases de données de la modélisation au SQL, Laurent Audibert – Collection Ellipses, 2009
et « Bases de donnée, Georges Gardarin, Eyrolles, 2003 »

Sélection



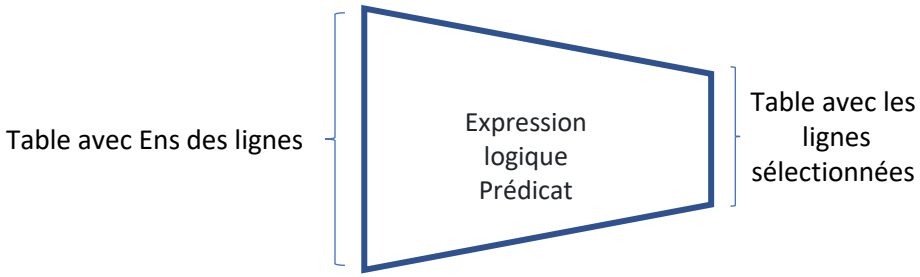
La sélection (parfois appelée Restriction)

Génère une relation regroupant exclusivement toutes les occurrences de la relation R qui satisfont l'expression logique E

Notation : $\sigma_{(E)}R$

Il s'agit d'une opération unaire essentielle dont la signature est :

relation x expression logique → relation



La sélection permet de choisir (*i.e.* sélectionner) des lignes dans le tableau.

Le résultat de la sélection est donc une nouvelle relation qui a les mêmes attributs que *R* avec moins de lignes.

Si *R* est vide (*i.e.* ne contient aucune occurrence), la relation qui résulte de la sélection est vide.

SQL :

L'opérateur de sélection $\sigma_{(prédicat)}(relation)$ se traduit tout simplement en SQL par la requête :

```
SELECT * FROM relation WHERE prédicat
```

Sélection

La sélection (parfois appelée Restriction)

Exemple : $\sigma_{(\text{Numéro} \geq 5)}\text{Personne}$

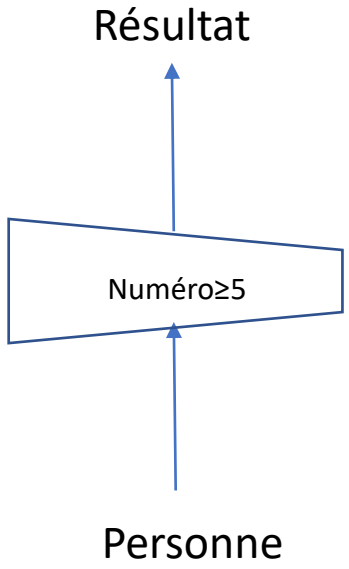
Personne		
Numéro	**Nom**	**Prénom**
5	Durand	Caroline
1	Germain	Stan
12	Dupont	Lisa
3	Germain	Rose-Marie

Sélection

La sélection (parfois appelée Restriction)

Exemple : $\sigma_{(\text{Numéro} \geq 5)}\text{Personne}$

Personne		
Numéro	**Nom**	**Prénom**
5	Durand	Caroline
1	Germain	Stan
12	Dupont	Lisa
3	Germain	Rose-Marie



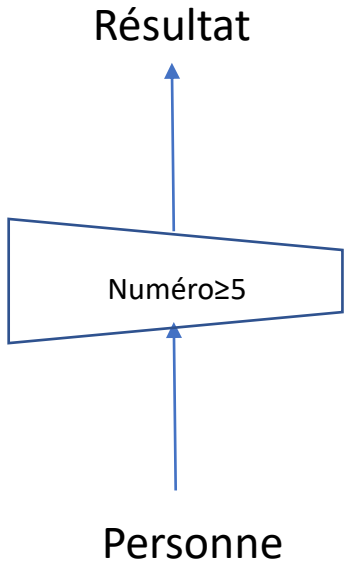
Sélection

La sélection (parfois appelée Restriction)

Exemple : $\sigma_{(\text{Numéro} \geq 5)}\text{Personne}$

```
SELECT * FROM Personne WHERE Numéro ≥ 5
```

Personne		
Numéro	**Nom**	**Prénom**
5	Durand	Caroline
1	Germain	Stan
12	Dupont	Lisa
3	Germain	Rose-Marie



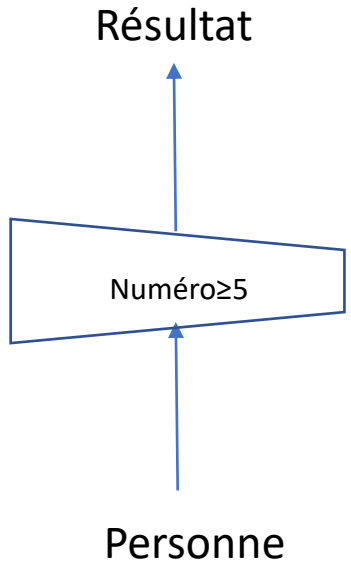
Sélection

La sélection (parfois appelée Restriction)

Exemple : $\sigma_{(\text{Numéro} \geq 5)}\text{Personne}$

```
SELECT * FROM Personne WHERE Numéro ≥ 5
```

Personne		
Numéro	**Nom**	**Prénom**
5	Durand	Caroline
1	Germain	Stan
12	Dupont	Lisa
3	Germain	Rose-Marie



$\sigma_{(\text{Numéro} \geq 5)}\text{Personne}$		
Numéro	**Nom**	**Prénom**
5	Durand	Caroline
12	Dupont	Lisa

Sélection

La sélection (parfois appelée Restriction)

Exemple : $\sigma_{(Mill>1983)}VINS$

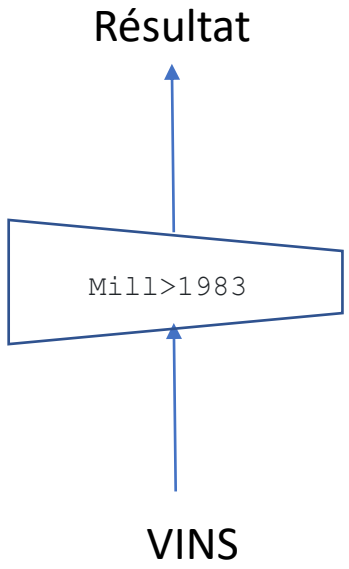
VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C

Sélection

La sélection (parfois appelée Restriction)

Exemple : $\sigma_{(Mill>1983)}VINS$

VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C



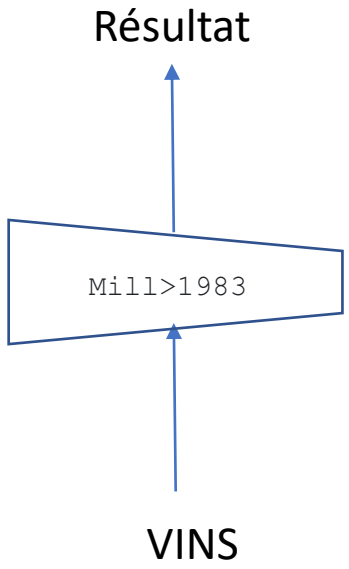
Sélection

La sélection (parfois appelée Restriction)

Exemple : $\sigma_{(Mill>1983)}VINS$

```
SELECT * FROM VINS WHERE Mill>1983
```

VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C

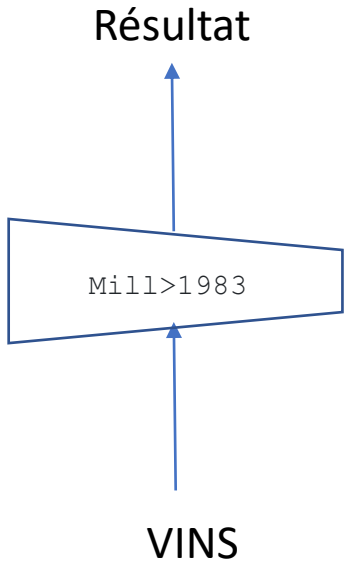


Sélection

La sélection (parfois appelée Restriction)

Exemple : $\sigma_{(Mill>1983)}VINS$

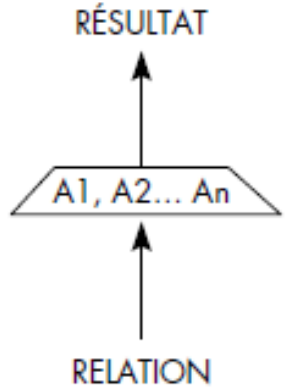
```
SELECT * FROM VINS WHERE Mill>1983
```



VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C

VINS	Cru	Mill	Région	Qualité
	JULIENAS	1986	BEAUJOLAIS	C

Projection



La projection

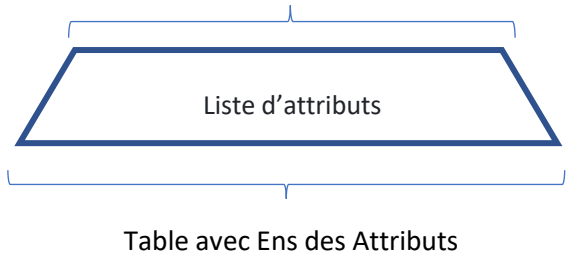
La projection consiste à :

- supprimer les attributs autres que A_1,… A_n d'une relation
- éliminer les n-uplets en double apparaissant dans la nouvelle relation ;

Notation : $\Pi_{(A1\ldots\ An)}R$

Il s'agit d'une opération unaire essentielle dont la signature est :

relation x liste d'attributs → relation



La projection permet de choisir des colonnes dans le tableau.

Si *R* est vide, la relation qui résulte de la projection est vide, mais pas forcément équivalente (elle contient généralement moins d'attributs).

SQL :

L'opérateur de projection $\Pi_{(A1,\ldots\ An)}$(*relation*) se traduit tout simplement en SQL par la requête :

```
SELECT DISTINCT A_1, ..., A_n FROM relation
```

Projection

La projection

Exemple : $\Pi_{(Nom)}$Personne

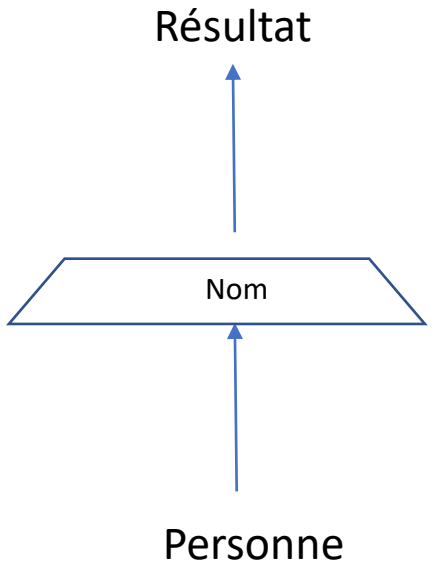
Personne		
Numéro	**Nom**	**Prénom**
5	Durand	Caroline
1	Germain	Stan
12	Dupont	Lisa
3	Germain	Rose-Marie

Projection

La projection

Exemple : $\Pi_{(Nom)}$Personne

Personne		
Numéro	**Nom**	**Prénom**
5	Durand	Caroline
1	Germain	Stan
12	Dupont	Lisa
3	Germain	Rose-Marie



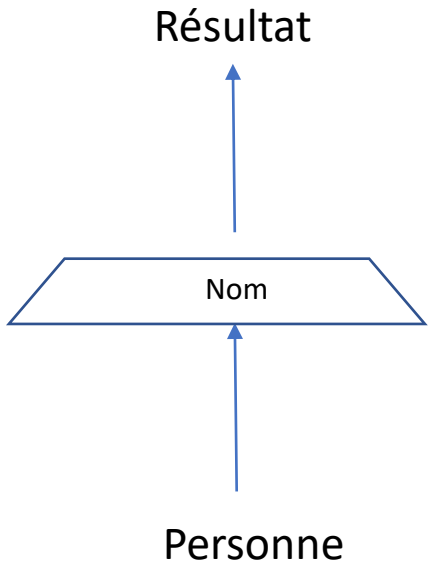
Projection

La projection

Exemple : $\Pi_{(Nom)}$Personne

```
SELECT DISTINCT Nom FROM Personne
```

Personne		
Numéro	**Nom**	**Prénom**
5	Durand	Caroline
1	Germain	Stan
12	Dupont	Lisa
3	Germain	Rose-Marie



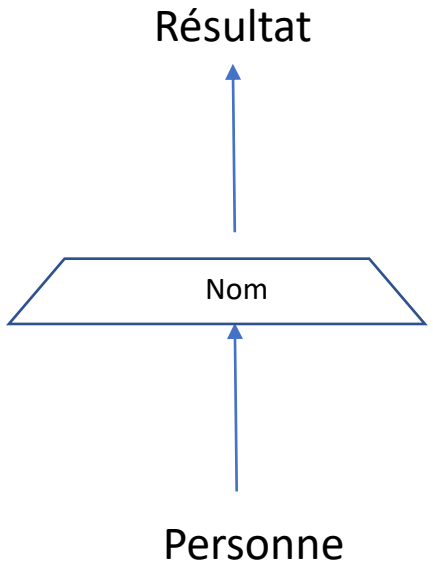
Projection

La projection

Exemple : $\Pi_{(Nom)}$Personne

```
SELECT DISTINCT Nom FROM Personne
```

Personne		
Numéro	**Nom**	**Prénom**
5	Durand	Caroline
1	Germain	Stan
12	Dupont	Lisa
3	Germain	Rose-Marie



$\Pi_{(Nom)}$Personne
Nom
Durand
Germain
Dupont

Projection

La projection

Exemple : $\Pi_{(Cru,\ Région)}Vins$

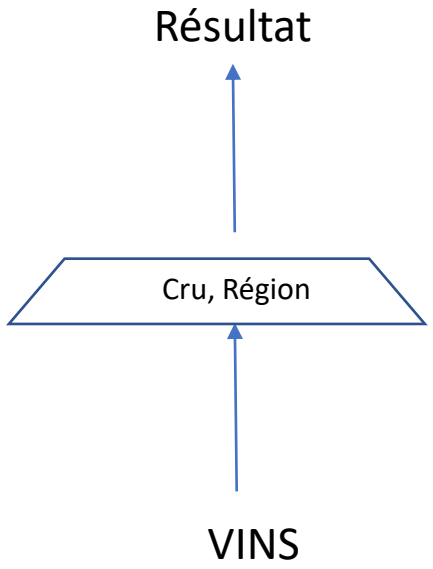
VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C

Projection

La projection

Exemple : $\Pi_{(Cru,\ Région)}Vins$

VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C



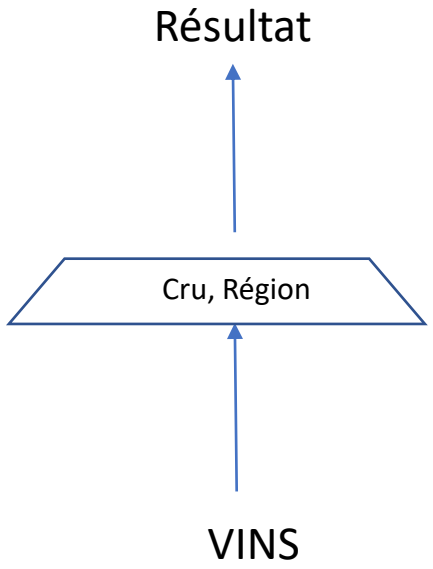
Projection

La projection

Exemple : $\Pi_{(Cru,\ Région)}Vins$

```
SELECT DISTINCT Cru, Région FROM VINS
```

VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C

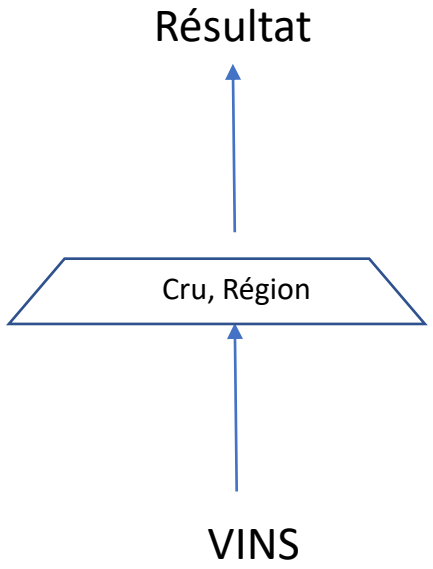


Projection

La projection

Exemple : $\Pi_{(Cru,\ Région)}Vins$

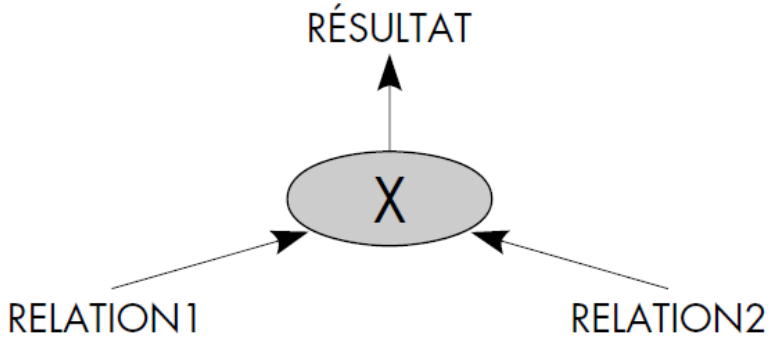
```
SELECT DISTINCT Cru, Région FROM VINS
```



VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C

Cru	Région
VOLNAY	BOURGOGNE
CHENAS	BEAUJOLAIS
JULIENAS	BEAUJOLAIS

Produit cartésien



Le produit cartésien

Le produit cartésien est une opération portant sur deux relations R_1 et R_2
construit une troisième relation regroupant exclusivement toutes les possibilités de combinaison des occurrences des relations R_1 et

Notation : $R_1 \times R_2$

Il s'agit d'une opération binaire commutative essentielle dont la signature est :

relation x relation → relation

- Le résultat du produit cartésien est une nouvelle relation qui a tous les attributs de R_1 et tous ceux de R_2.
- Si R_1 ou R_2 ou les deux sont vides, la relation qui résulte du produit cartésien est vide.
- Le nombre d'occurrences de la relation qui résulte du produit cartésien est le nombre d'occurrences de R_1 multiplié par le nombre d'occurrences de R_2.

SQL :

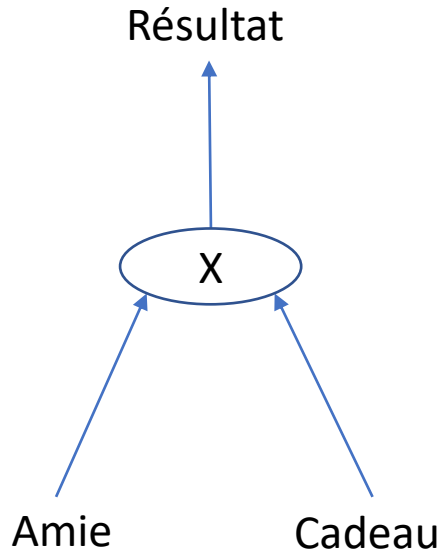
L'opérateur de produit cartésien relation1 × relation2 se traduit en SQL par la requête :

```
SELECT * FROM relation_1, relation_2
```

Produit cartésien

Le produit cartésien

```
SELECT * FROM Amie, Cadeau
```



R = Amie × Cadeau

Relation *Amie*

Nom	Prénom
Fourt	Lisa
Juny	Carole

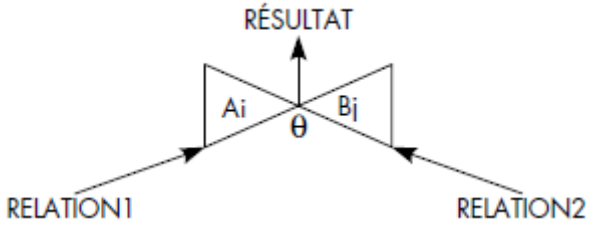
Relation *Cadeau*

Article	Prix
livre	45
poupée	25
montre	87

Relation *R*

Nom	Prénom	Article	Prix
Fourt	Lisa	livre	45
Fourt	Lisa	poupée	25
Fourt	Lisa	montre	87
Juny	Carole	livre	45
Juny	Carole	poupée	25
Juny	Carole	montre	87

Jointure



La jointure

La jointure est une opération portant sur deux relations R_1 et R_2

construit une troisième relation regroupant exclusivement toutes les possibilités de combinaison des occurrences des relations R_1 et R_2

qui satisfont l'expression logique E.

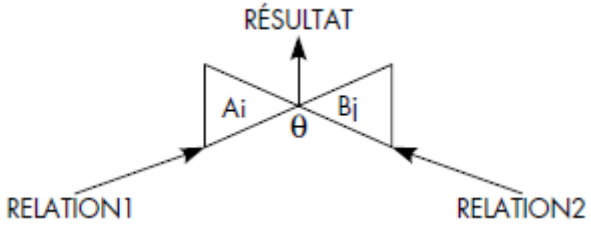
Notation : $R_1 \bowtie_E R_2$

Il s'agit d'une opération binaire commutative dont la signature est :

relation x relation x expression logique → relation

- Si R_1 ou R_2 ou les deux sont vides, la relation qui résulte de la jointure est vide.

Jointure



La jointure

- Elle peut être vue comme une extension du produit cartésien avec une condition permettant de comparer des attributs
- En fait, la jointure n'est rien d'autre qu'un produit cartésien suivi d'une sélection :

$$R_1 \bowtie_E R_2 = \sigma_{(E)}(R_1 \times R_2)$$

- On peut également dire que le produit cartésien n'est rien d'autre qu'une jointure dans laquelle l'expression logique E est toujours vraie : $R_1 \times R_2 = R_1 \bowtie_{true} R_2$

SQL :

```
SELECT * FROM relation_1, relation_2
```
Produit cartésien

```
SELECT * FROM relation_1 JOIN relation_2
```
sans condition, **JOIN** est équivalent à un produit cartésien (produit croisé) : tous les couples de tuples possibles sont présents dans le résultat.

```
SELECT * FROM relation_1 CROSS JOIN relation_2
```

CROSS JOIN permet d'indiquer explicitement que l'on cherche à faire un produit cartésien

Jointure

La jointure (avec une condition toujours vérifiée (vrais))

R = Famille ⋈((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

R = Famille ⋈ ((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

Relation *Famille*

Nom	Prénom	Age
Fourt	Lisa	6
Juny	Carole	42
Fidus	Laure	16

Relation *Cadeau*

AgeC	Article	Prix
99	livre	30
6	poupée	60
20	baladeur	45
10	déguisement	15

Jointure

La jointure (avec une condition toujours vérifiée (vrais))

R = Famille ⋈((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

R = Famille ⋈ ((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

Relation *Famille*

Nom	Prénom	Age
Fourt	Lisa	6
Juny	Carole	42
Fidus	Laure	16

Relation *Cadeau*

AgeC	Article	Prix
99	livre	30
6	poupée	60
20	baladeur	45
10	déguisement	15

Relation *R*

Nom	Prénom	Age	AgeC	Article	Prix
Fourt	Lisa	6	99	livre	30
Fourt	Lisa	6	20	baladeur	45
Fourt	Lisa	6	10	déguisement	15
Juny	Carole	42	99	livre	30
Fidus	Laure	16	99	livre	30
Fidus	Laure	16	20	baladeur	45

Jointure

La jointure (avec une condition toujours vérifiée (vrais))

R = Famille ⋈((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

Age ≤ AgeC

Prix < 50

R = Famille ⋈ ((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

Relation *Famille*

Nom	Prénom	Age
Fourt	Lisa	6
Juny	Carole	42
Fidus	Laure	16

Relation *Cadeau*

AgeC	Article	Prix
99	livre	30
6	poupée	60
20	baladeur	45
10	déguisement	15

Relation *R*

Nom	Prénom	Age	AgeC	Article	Prix
Fourt	Lisa	6	99	livre	30
Fourt	Lisa	6	20	baladeur	45
Fourt	Lisa	6	10	déguisement	15
Juny	Carole	42	99	livre	30
Fidus	Laure	16	99	livre	30
Fidus	Laure	16	20	baladeur	45

Jointure

La jointure (avec une condition toujours vérifiée (vrais))

R = Famille ⋈((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

R = Famille ⋈ ((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

Relation *Famille*

Nom	Prénom	Age
Fourt	Lisa	6
Juny	Carole	42
Fidus	Laure	16

Relation *Cadeau*

AgeC	Article	Prix
99	livre	30
6	poupée	60
20	baladeur	45
10	déguisement	15

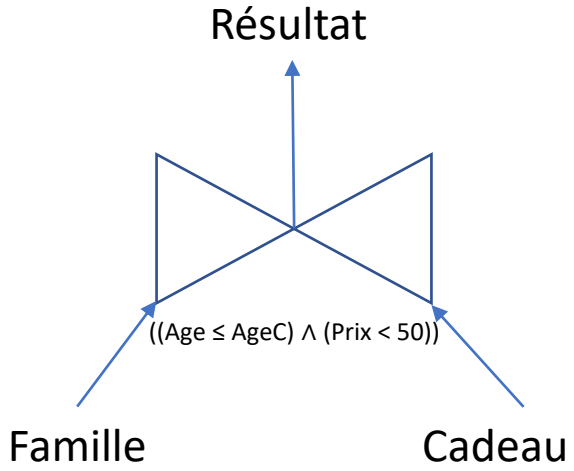
Relation *R*

Nom	Prénom	Age	AgeC	Article	Prix
Fourt	Lisa	6	99	livre	30
Fourt	Lisa	6	20	baladeur	45
Fourt	Lisa	6	10	déguisement	15
Juny	Carole	42	99	livre	30
Fidus	Laure	16	99	livre	30
Fidus	Laure	16	20	baladeur	45

Jointure

La jointure (avec une condition toujours vérifiée (vrais))

R = Famille ⋈((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau



R = Famille ⋈ ((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

Relation *Famille*

Nom	Prénom	Age
Fourt	Lisa	6
Juny	Carole	42
Fidus	Laure	16

Relation *Cadeau*

AgeC	Article	Prix
99	livre	30
6	poupée	60
20	baladeur	45
10	déguisement	15

Relation *R*

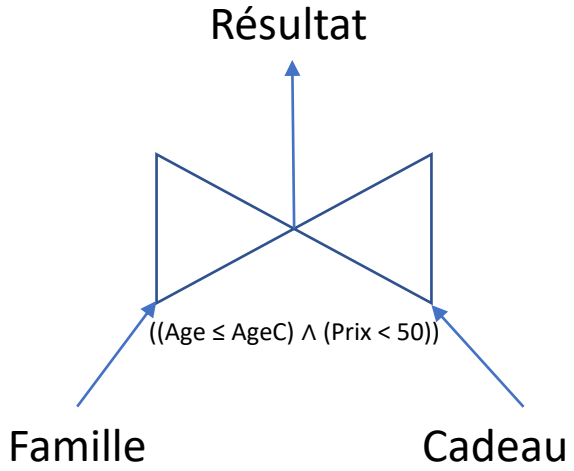
Nom	Prénom	Age	AgeC	Article	Prix
Fourt	Lisa	6	99	livre	30
Fourt	Lisa	6	20	baladeur	45
Fourt	Lisa	6	10	déguisement	15
Juny	Carole	42	99	livre	30
Fidus	Laure	16	99	livre	30
Fidus	Laure	16	20	baladeur	45

Jointure

La jointure (avec une condition toujours vérifiée (vrais))

$R = Famille \bowtie ((Age \leq AgeC) \wedge (Prix < 50))\ Cadeau$

```
SELECT * FROM Famille JOIN Cadeau ON
        Famille.Age≤Cadeau.AgeC AND Cadeau.Prix<50
```



$R = Famille \bowtie ((Age \leq AgeC) \wedge (Prix < 50))\ Cadeau$

Relation *Famille*

Nom	Prénom	Age
Fourt	Lisa	6
Juny	Carole	42
Fidus	Laure	16

Relation *Cadeau*

AgeC	Article	Prix
99	livre	30
6	poupée	60
20	baladeur	45
10	déguisement	15

Relation *R*

Nom	Prénom	Age	AgeC	Article	Prix
Fourt	Lisa	6	99	livre	30
Fourt	Lisa	6	20	baladeur	45
Fourt	Lisa	6	10	déguisement	15
Juny	Carole	42	99	livre	30
Fidus	Laure	16	99	livre	30
Fidus	Laure	16	20	baladeur	45

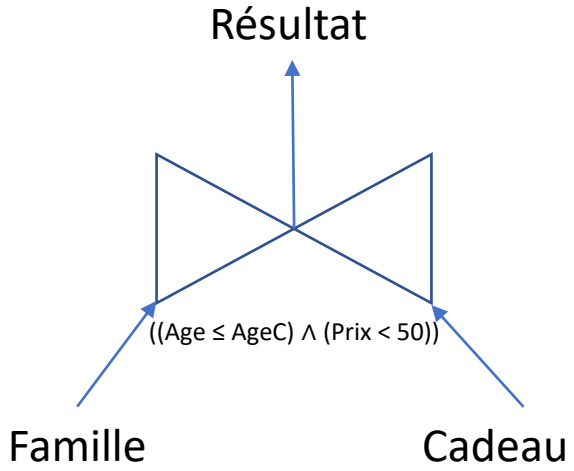
Jointure

La jointure (avec une condition toujours vérifiée (vrais))

R = Famille ⋈((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

```
SELECT * FROM Famille JOIN Cadeau ON
        Famille.Age≤Cadeau.AgeC AND Cadeau.Prix<50

SELECT * FROM Famille, Cadeau
  WHERE Famille.Age≤Cadeau.AgeC AND Cadeau.Prix<50
```



R = Famille ⋈ ((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

Relation *Famille*

Nom	Prénom	Age
Fourt	Lisa	6
Juny	Carole	42
Fidus	Laure	16

Relation *Cadeau*

AgeC	Article	Prix
99	livre	30
6	poupée	60
20	baladeur	45
10	déguisement	15

Relation *R*

Nom	Prénom	Age	AgeC	Article	Prix
Fourt	Lisa	6	99	livre	30
Fourt	Lisa	6	20	baladeur	45
Fourt	Lisa	6	10	déguisement	15
Juny	Carole	42	99	livre	30
Fidus	Laure	16	99	livre	30
Fidus	Laure	16	20	baladeur	45

Jointure

La jointure (avec une condition toujours vérifiée (vrais))

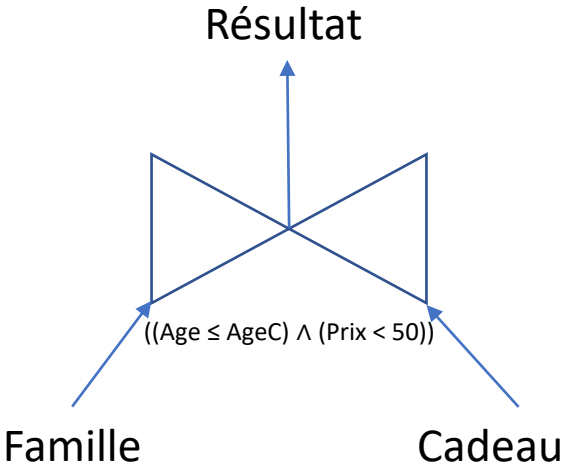
R = Famille ⋈((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

```
SELECT * FROM Famille JOIN Cadeau ON
        Famille.Age≤Cadeau.AgeC AND Cadeau.Prix<50
```

SQL2

```
SELECT * FROM Famille, Cadeau
  WHERE Famille.Age≤Cadeau.AgeC AND Cadeau.Prix<50
```

SQL1



R = Famille ⋈ ((Age ≤ AgeC) ∧ (Prix < 50)) Cadeau

Relation *Famille*

Nom	Prénom	Age
Fourt	Lisa	6
Juny	Carole	42
Fidus	Laure	16

Relation *Cadeau*

AgeC	Article	Prix
99	livre	30
6	poupée	60
20	baladeur	45
10	déguisement	15

Relation *R*

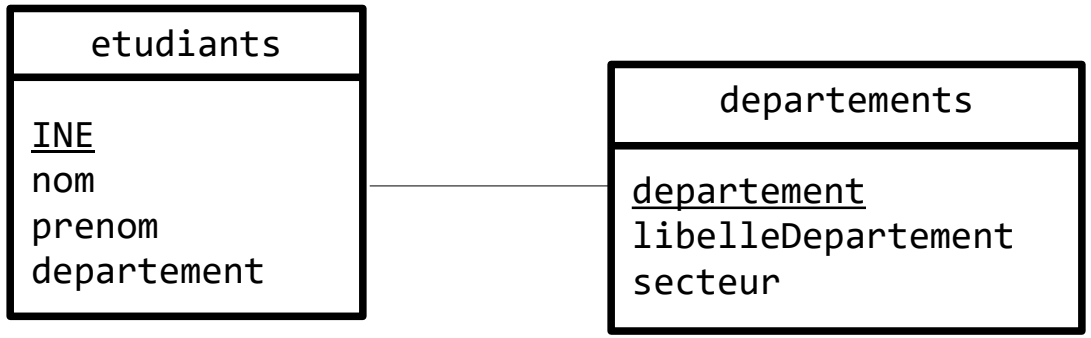
Nom	Prénom	Age	AgeC	Article	Prix
Fourt	Lisa	6	99	livre	30
Fourt	Lisa	6	20	baladeur	45
Fourt	Lisa	6	10	déguisement	15
Juny	Carole	42	99	livre	30
Fidus	Laure	16	99	livre	30
Fidus	Laure	16	20	baladeur	45

SQL

Syntaxe SQL2

Cas des jointures
Variations sur la syntaxe

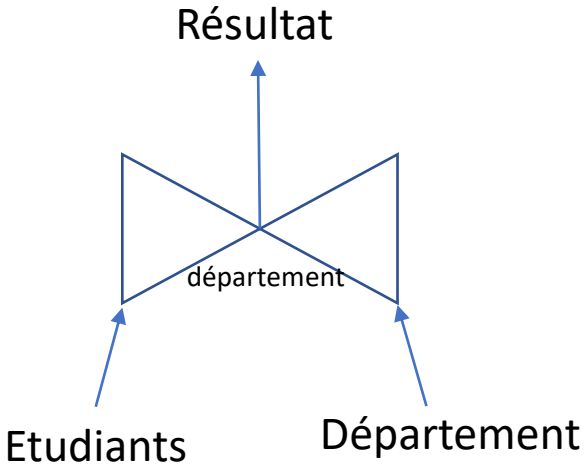
Exemple de situation



INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

Jointure naturelle



`etudiants`

INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

`departements`

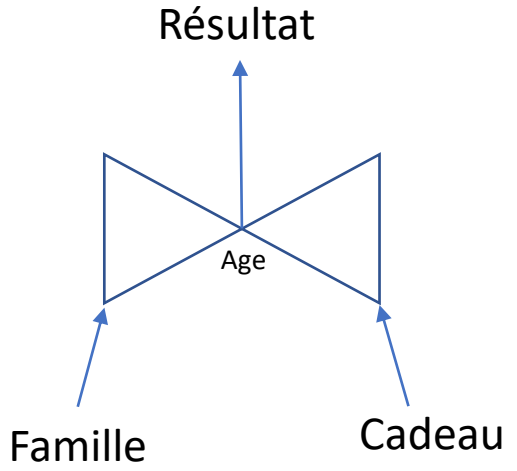
departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

`etudiants NATURAL JOIN departements`

departement	INE	nom	prenom	libelleDepartement	secteur
SD	1	Ah	Alice	Science de données	tertiaire
INFO	2	Beh	Bob	Informatique	secondaire
SD	3	Ceh	Charlène	Science de données	tertiaire

- On joint 2 tuples ssi ***tous*** les attributs de même nom dans les deux tables ont les mêmes valeurs
- Les attributs de jointure n'apparaissent qu'une fois, en début de relation
- Ici, il n y a que « département » qui est en commun, il apparait au début du résultat

Jointure naturelle



La jointure naturelle

```
Famille NATURAL JOIN Cadeau
```

R = Famille ⋈ Cadeau ou R = Famille $\bowtie_{Age}$ Cadeau

Relation *Famille*

Nom	Prénom	Age
Fourt	Lisa	6
Juny	Carole	40
Fidus	Laure	20
Choupy	Emma	6

Relation *Cadeau*

Age	Article	Prix
40	livre	45
6	poupée	25
20	montre	87

Relation *R*

Age	Nom	Prénom	Article	Prix
6	Fourt	Lisa	poupée	25
40	Juny	Carole	livre	45
20	Fidus	Laure	montre	87
6	Choupy	Emma	poupée	25

Jointure naturelle

```
VINS NATURAL JOIN LOCALISATION
```

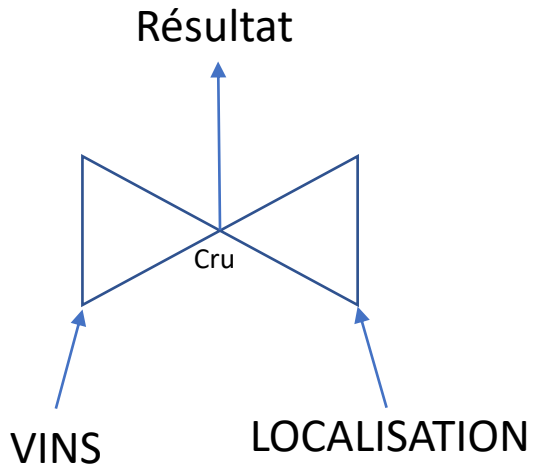
VINS	Cru	Mill	Qualité
	VOLNAY	1983	A
	VOLNAY	1979	B
	CHABLIS	1983	A
	JULIENAS	1986	C

⋈

LOCALISATION	Cru	Région	QualMoy
	VOLNAY	BOURGOGNE	A
	CHABLIS	BOURGOGNE	A
	CHABLIS	CALIFORNIE	B

↓

VINSREG	Cru	Mill	Qualité	Région	QualMoy
	VOLNAY	1983	A	BOURGOGNE	A
	VOLNAY	1979	B	BOURGOGNE	A
	CHABLIS	1983	A	BOURGOGNE	A
	CHABLIS	1983	A	CALIFORNIE	B



Jointure naturelle

La jointure naturelle

Une jointure naturelle est une jointure dans laquelle l'expression logique E est un test d'égalité entre les attributs qui portent le même nom dans les relations R_1 et R_2

Dans la relation construite, ces attributs ne sont pas dupliqués, mais fusionnés en une seule colonne par couple d'attributs, qui apparait au début.

Notation : $R_1 \bowtie R_2$

On peut préciser explicitement les attributs communs à R_1 et R_2 sur lesquels porte la jointure :

Notation : $R_1 \bowtie_{A1,\dots An} R_2$

Jointure naturelle

La jointure naturelle

Généralement, R_1 et R_2 n'ont qu'un attribut en commun.

Dans ce cas, une jointure naturelle est équivalente à une *équijointure* dans laquelle l'attribut de R_1 et celui de R_2 sont justement les deux attributs qui portent le même nom.

Lorsque l'on désire effectuer une jointure naturelle entre R_1 et R_2 sur un attribut A_1 commun à R_1 et R_2,

Il vaut mieux écrire $R_1 \bowtie_{A1} R_2$ que $R_1 \bowtie R_2$

si R_1 et R_2 possèdent deux attributs portant un nom commun, A_1 et A_2,

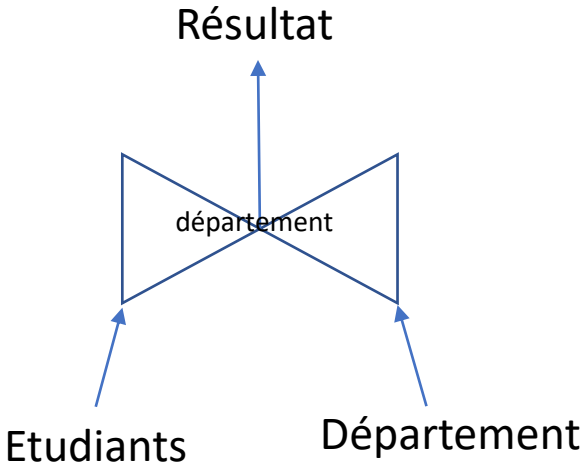
$R_1 \bowtie_{A1} R_2$ est bien une jointure naturelle sur l'attribut A_1,

Mais $R_1 \bowtie R_2$ est une jointure naturelle sur le couple d'attributs A_1, A_2,

Ce qui produit un résultat très différent !

→ Quand on souhaite faire une jointure en conditionnant sur un attribut il est recommandé d'utiliser

JOIN …. USING



etudiants

INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departements

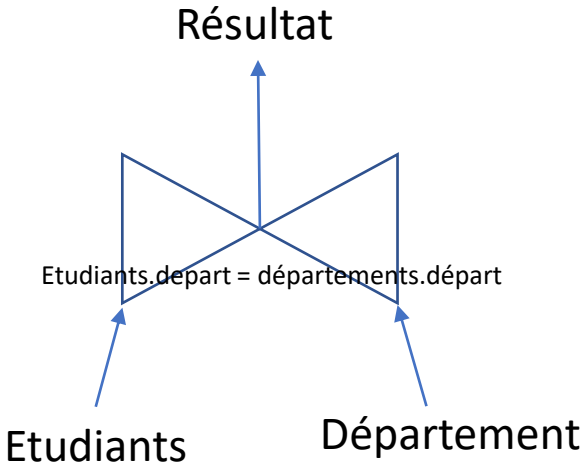
departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

```
etudiants JOIN departements USING(departement)
```

departement	INE	nom	prenom	libelleDepartement	secteur
SD	1	Ah	Alice	Science de données	tertiaire
INFO	2	Beh	Bob	Informatique	secondaire
SD	3	Ceh	Charlène	Science de données	tertiaire

- On joint 2 tuples ssi les attributs de même nom apparaissant dans **USING** ont les mêmes valeurs
- Les attributs de jointure n'apparaissent qu'une fois, en début de relation

JOIN .... ON



etudiants

INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departements

departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

```
etudiants JOIN departements
ON etudiants.departement = departements.departement
```

INE	nom	prenom	departement	departement	libelleDepartement	secteur
1	Ah	Alice	SD	SD	Science de données	tertiaire
2	Beh	Bob	INFO	INFO	Informatique	secondaire
3	Ceh	Charlène	SD	SD	Science de données	tertiaire

- On joint 2 tuples ssi les conditions de la clause **ON** sont satisfaites (vrais).
- On peut utiliser autre chose que l'égalité. Les attributs peuvent avoir des noms différents.

- *Les attributs de jointure en provenance des deux tables apparaissent deux fois*

JOIN .... ON

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

communes

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	
56260	Vannes	56	
58194	Nevers	58	

departements

num Departement	libelle Departement	commune ChefLieu	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

JOIN .... ON

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

communes

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	
56260	Vannes	56	
58194	Nevers	58	

departements

num Departement	libelle Departement	commune ChefLieu	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

JOIN .... ON

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

communes

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	
56260	Vannes	56	
58194	Nevers	58	

departements

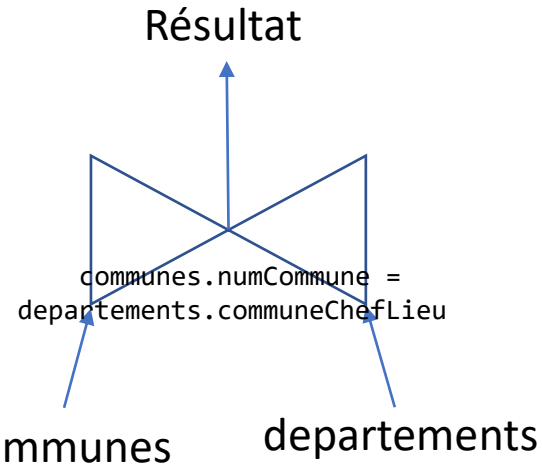
num Departement	libelle Departement	commune ChefLieu	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

JOIN .... ON

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]



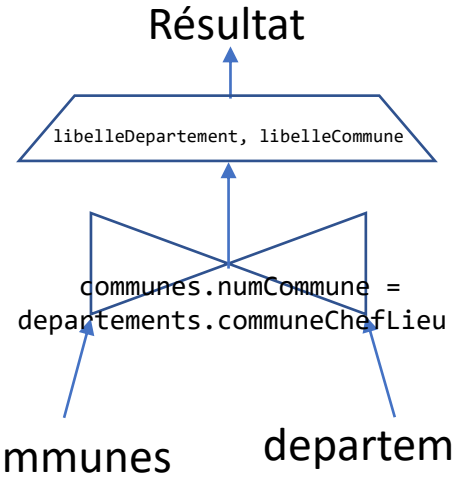
communes

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	
56260	Vannes	56	
58194	Nevers	58	

departements

num Departement	libelle Departement	commune ChefLieu	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

JOIN .... ON



communes(<u>numCommune</u>, libelleCommune, #numDepartement, population) ;

departements(<u>numDepartement</u>, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

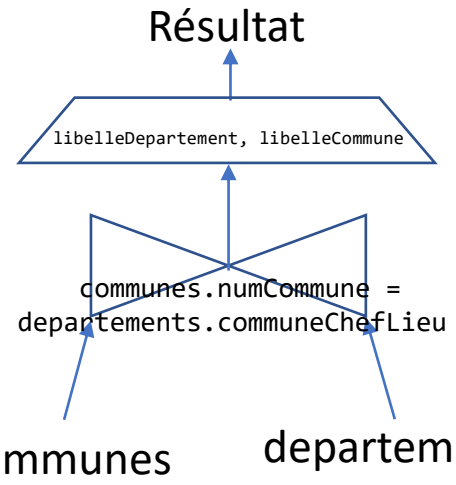
communes

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	
56260	Vannes	56	
58194	Nevers	58	

departements

num Departement	libelle Departement	commune ChefLieu	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

JOIN .... ON



communes(<u>numCommune</u>, libelleCommune, #numDepartement, population) ;

departements(<u>numDepartement</u>, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

communes

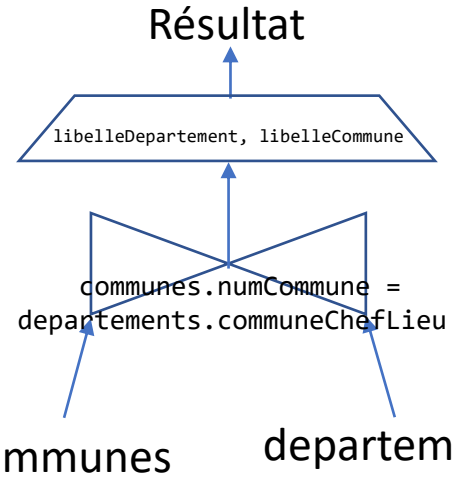
num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	
56260	Vannes	56	
58194	Nevers	58	

departements

num Departement	libelle Departement	commune ChefLieu	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

```
SELECT
 libelleDepartement, libelleCommune AS chefLieu
FROM
 communes JOIN departements
ON
 communes.numCommune = departements.communeChefLieu ;
```

JOIN .... ON



communes(<u>numCommune</u>, libelleCommune, #numDepartement, population) ;

departements(<u>numDepartement</u>, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

communes

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	
56260	Vannes	56	
58194	Nevers	58	

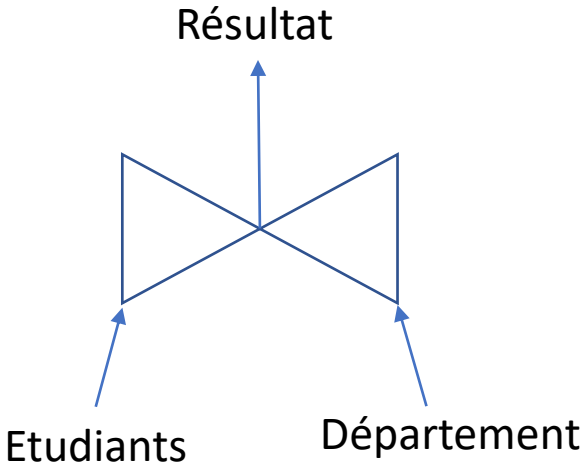
departements

num Departement	libelle Departement	commune ChefLieu	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

```
SELECT
 libelleDepartement, libelleCommune AS chefLieu
FROM
 communes JOIN departements
ON
 communes.numCommune = departements.communeChefLieu ;
```

Recommandation : A utiliser si les attributs n'ont pas le même nom ou bien sur non égalité

JOIN (sans condition)



etudiants

INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departements

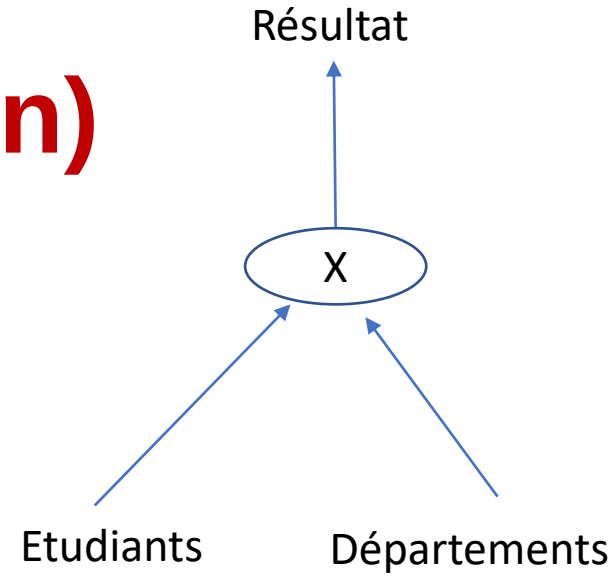
departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

etudiants **JOIN** departements

INE	nom	prenom	departement	departement	libelleDepartement	secteur
1	Ah	Alice	SD	SD	Science de données	tertiaire
2	Beh	Bob	INFO	SD	Science de données	tertiaire
3	Ceh	Charlène	SD	SD	Science de données	tertiaire
1	Ah	Alice	SD	INFO	Informatique	secondaire
2	Beh	Bob	INFO	INFO	Informatique	secondaire
3	Ceh	Charlène	SD	INFO	Informatique	secondaire

- sans condition, **JOIN** est équivalent à un produit cartésien (produit croisé) :
- tous les couples de tuples possibles sont présents dans le résultat

CROSS JOIN (sans condition)



etudiants

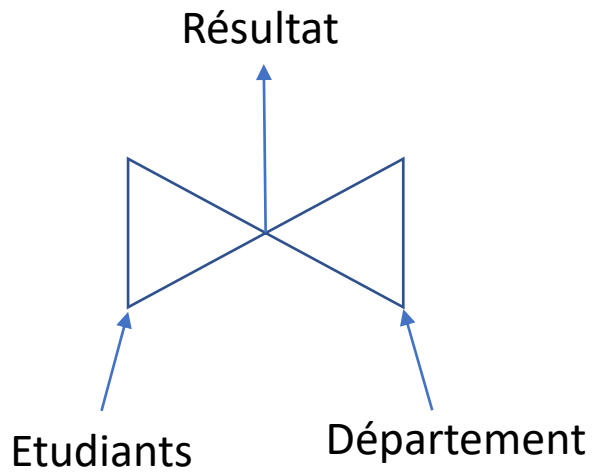
INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departements

departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

etudiants **CROSS JOIN** departements

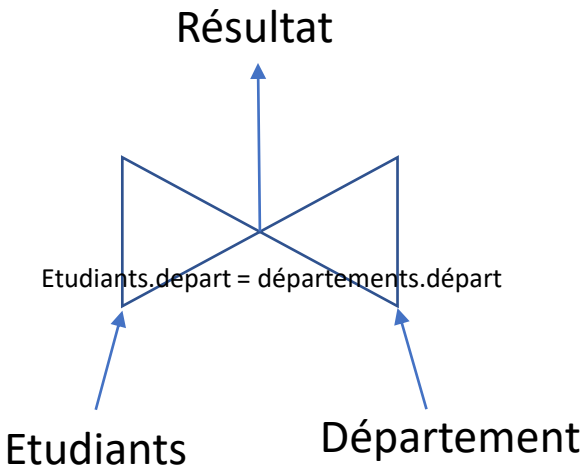
INE	nom	prenom	departement	departement	libelleDepartement	secteur
1	Ah	Alice	SD	SD	Science de données	tertiaire
2	Beh	Bob	INFO	SD	Science de données	tertiaire
3	Ceh	Charlène	SD	SD	Science de données	tertiaire
1	Ah	Alice	SD	INFO	Informatique	secondaire
2	Beh	Bob	INFO	INFO	Informatique	secondaire
3	Ceh	Charlène	SD	INFO	Informatique	secondaire



CROSS JOIN

permet d'indiquer explicitement que l'on cherche à faire un produit cartésien

Syntaxe SQL1 (virgule et WHERE)



etudiants

INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departements

departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

```
SELECT
 *
FROM
 etudiants, departements
WHERE
 etudiants.departement = departements.departement ;
```

INE	nom	prenom	departement	departement	libelleDepartement	secteur
1	Ah	Alice	SD	SD	Science de données	tertiaire
2	Beh	Bob	INFO	INFO	Informatique	secondaire
3	Ceh	Charlène	SD	SD	Science de données	tertiaire

SQL
un peu plus loin

Introduction

Le langage SQL (*Structured Query Language*)

- Langage d'accès normalisé aux bases de données.
- Il est aujourd'hui supporté par la plupart des produits commerciaux que ce soit par les systèmes de gestion de bases de données micro tel que *Access* ou par les produits plus professionnels tels que *Oracle*.
- Il a fait l'objet de plusieurs normes ANSI/ISO dont la plus répandue aujourd'hui est la norme SQL2 qui a été définie en 1992.
- Le succès du langage SQL est dû essentiellement à sa simplicité et au fait qu'il s'appuie sur le schéma conceptuel pour énoncer des requêtes en laissant le SGBD responsable de la stratégie d'exécution.
- Le langage SQL propose un langage de requêtes.
- Le langage SQL ne possède pas la puissance d'un langage de programmation : entrées/sorties, instructions conditionnelles, boucles et affectations. Pour certains traitements il est donc nécessaire de coupler le langage SQL avec un langage de programmation plus complet.

Introduction

Le langage SQL (*Structured Query Language*)

Définition

SQL est un **langage relationnel**,

il manipule donc des **tables** (*i.e.* des relations, c'est-à-dire des ensembles),

par l'intermédiaire de **requêtes**

qui produisent également des **tables**.

Introduction

Le langage SQL (*Structured Query Language*)

Les instructions SQL sont regroupées en catégories en fonction de leur utilité et des entités manipulées.

Nous pouvons distinguer cinq catégories, qui permettent :

1. La définition des éléments d'une base de données (tables, colonnes, clés, index, contraintes…),
2. La manipulation des données (insertion, suppression, modification, extraction…),
3. La gestion des droits d'accès aux données (acquisition et révocation des droits),
4. La gestion des transactions,
5. Le SQL intégré.

Introduction

Langage SQL

Structured Query Language

DDL
Data Definition Language

DCL
Data Control Language

DML
Data Manipulation Language

TCL
Transaction Control Language

Introduction

Langage SQL

Structured Query Language

DDL

Data Definition Language

Définir les objets de la base de données : **Tables**, **Vues**, **Déclencheurs**, **Procédures** & **Fonctions** stockées.

Les instructions du LDD sont : CREATE, ALTER, DROP, AUDIT, NOAUDIT, ANALYZE, RENAME, TRUNCATE.

Introduction

Langage SQL

Structured Query Language

DML
Data Manipulation Language

Manipulation des données de la base : ***Sélectionner*** (obtenir),
ajouter, ***supprimer*** ou ***modifier*** des tuples

Les instructions du LMD sont : INSERT, UPDATE, DELETE, SELECT, EXPLAIN, PLAN, LOCK TABLE.

Introduction

Langage SQL

Structured Query Language

Qui a le droit de faire quelle opération sur les objets de la base de données ? (tables, vues, attributs, procédures & fonctions stockées, …)

DCL
Data Control Language

Les instructions du DCL sont : GRANT, REVOKE.

Introduction

Langage SQL

Structured Query Language

Gestion des transactions (implémentation des propriétés ACID ; **Atomicité**, **Cohérence**, **Isolation**, **Durabilité**)

Les instructions du TCL sont : COMMIT, SAVEPOINT, ROLLBACK, SET TRANSACTION

TCL

Transaction Control Language

Introduction

Langage SQL

Structured Query Language

Le **SQL intégré** (*Embedded SQL*)

permet d'utiliser SQL dans un langage de troisième génération (C, Java, Cobol, etc.) :

- déclaration d'objets ou d'instructions ;
- exécution d'instructions ;
- gestion des variables et des curseurs ;
- traitement des erreurs.

Les instructions du SQL intégré sont : DECLARE, TYPE, DESCRIBE, VAR, CONNECT, PREPARE, EXECUTE, OPEN, FETCH, CLOSE, WHENEVER.

Introduction

Langage SQL

Structured Query Language

DDL
Data Definition Language

DCL
Data Control Language

DML
Data Manipulation Language

TCL
Transaction Control Language

Introduction

Langage SQL

Structured Query Language

DDL
Data Definition Language

DCL
Data Control Language

DML
Data Manipulation Language

TCL
Transaction Control Language

Introduction

Langage SQL

Structured Query Language

DDL
Data Definition Language

DCL
Data Control Language

DML
Data Manipulation Language

TCL
Transaction Control Language

Langage SQL

SELECT…
(obtenir des tuples)

INSERT INTO …
(ajouter des tuples à une table)

UPDATE …
(modifier les valeurs d'attributs de tuples)

DELETE FROM …
(supprimer des tuples)

DML
Data Manipulation Language

Manipulation de données

Généralités – littéraux textes

- Les ***littéraux textes*** sont entourés de ***guillemets simples*** (mais plusieurs SGBD permettent aussi l'usage de guillemets doubles).
- Si un guillemet simple apparaît dans le texte littéral, il faut l'échapper en le dédoublant.

Exemples :

- `'METZ'`
- `'DE NIRO'`
- `'l''éléphant'`

Manipulation de données

Généralités – littéraux dates et heures

Les ***littéraux temps*** sont donnés entre guillemets simples, dans l'ordre heure, minute, seconde et dixième de seconde.
Le séparateur de temps est le caractère deux-points ' : ', sauf pour les dixièmes de secondes (caractère point ' . ‘)

Exemples :

- '12:00:00.0’
- '18:00'

Les ***littéraux dates*** sont donnés entre guillemets simples, dans l'ordre année, mois, jour.
Le séparateur de temps est le caractère tiret ' - '

Exemples :

- '2012-02-28'
- '1999-11-30'

- **MS Access** ne suit pas la norme pour la gestion des dates & heures.
- En fonction de la configuration, **MySQL** permet d'utiliser des guillemets simples ou doubles pour tous les littéraux (texte, date, temps).

Manipulation de données

Généralités – littéraux numériques

- Les littéraux numériques n'ont pas à être entourés de guillemets.
- Ils peuvent utiliser les chiffres, les caractères + et -, un séparateur décimal (le caractère ' . ') et la notation e pour les puissances de 10.
- Les littéraux sans point décimal sont traités comme des entiers,
- Les littéraux avec point décimal sont traités comme des réels (flottant)

Exemples :

- `12` *(entier)*
- `12.0` *(réel)*
- `+12`
- `-3.5`
- `2e3` *(2000)*

Manipulation de données

Généralités – Syntaxe générale d'une requête

Une requête se termine par un point-virgule ';'

```
SELECT nom, prenom FROM etudiants ;
```

Les mots-clés du langage SQL sont insensibles à la casse.

```
SELECT nom, prenom FROM etudiants;
select nom, prenom from etudiants;
```

```
SELECT
        nom ,
        prenom
FROM
        etudiants;
```

Une requête peut être écrite sur une ou plusieurs lignes.
Des blancs (espaces & tabulations) peuvent être ajoutés pour améliorer la lisibilité de la requête sans en perturber le fonctionnement.

Manipulation de données

Généralités – Syntaxe générale d'une requête

SQL Permet de commenter son code :

- sous la forme de blocs de commentaires

```
/*
      Ceci est un bloc de commentaires.
*/
SELECT nom, prenom FROM etudiants;
```

- ou bien sous forme de commentaires de fin de lignes.

```
SELECT nom, prenom FROM etudiants; -- requête 1
select nom, prenom from etudiants; -- requête 2
```

Manipulation de données

Généralités – Sensibilité à la casse

Dans ***la norme***, ***SQL est insensible à la casse des noms de tables et d'attributs***. ça n'est pas le cas pour certaines configurations liées au SGBD (MySQL/mariaDB).

- Norme SQL :

etudiants(INE, nom, prenom, dateNaissance, sexe, ville, groupeTD, groupeTP)

=

ETUDIANTS(INE, NOM, PRENOM, DATENAISSANCE, SEXE, VILLE, GROUPETD, GROUPETP)

- mysql/mariaDB (par défaut) :

etudiants(INE, nom, prenom, dateNaissance, sexe, ville, groupeTD, groupeTP)

≠

ETUDIANTS(INE, NOM, PRENOM, DATENAISSANCE, SEXE, VILLE, GROUPETD, GROUPETP)

Manipulation de données

Structure générale d'une requête

SELECT

les attributs, littéraux ou combinaison/calculs à sélectionner, séparés par des virgules
"les colonnes que l'on cherche à obtenir"

FROM

une relation (éventuellement construite à partir d'autres relations par jointure ou sous-requête)
"la source des données à utiliser"

WHERE

Liste de prédicats, séparés par des connecteurs logiques (ET, OU, NON) permettant de filtrer (restreindre) les tuples à obtenir
"les conditions à vérifier pour qu'un tuple apparaisse dans le résultat"

GROUP BY

attributs à utiliser pour faire des groupes de tuples, en vu d'appliquer des fonctions d'agrégat
"les groupes de tuples à constituer pour les calculs statistiques"

HAVING

liste de prédicats, séparés par des connecteurs logiques (ET, OU, NON) , permettant de filtrer (restreindre) les tuples à obtenir depuis la relation après groupement
"les conditions à vérifier sur les indicateurs statistiques"

ORDER BY

liste des attributs servant à ordonner (trier) les tuples obtenus
"ce qui va servir à déterminer comment trier les tuples"

Manipulation de données

Quelques prédicats à connaitre

Connecteurs logiques, par niveau de priorité : **NOT**, **AND**, **OR**

Opération	opérateur	Remarque
inférieur à	`<`	
supérieur à	`>`	
inférieur ou égal à	`<=`	
supérieur ou égal à	`>=`	
égal à	`=`	
différent de	`<>` ou bien connecteur **NOT** sur un test d'égalité.	Par exemple, le prédicat `a<>b` est équivalent au prédicat `NOT a=b`
entre … et ...	**BETWEEN** … **AND** …	Par exemple `age BETWEEN 18 AND 25` est un prédicat équivalent à `age>=18 AND age<=25` mais se relit plus facilement
similaire à (valeurs textuelles)	**LIKE**	Peut s'utiliser avec des caractères jokers : `_` (underscore / souligné) permettant de remplacer n'importe quel caractère, et `%` (pourcentage) permettant de remplacer 0, 1 ou plusieurs caractères.
test de nullité	**IS NULL**	
test de non nullité	**IS NOT NULL** ou bien **NOT IS NULL**	

test d'apparition dans une liste	**IN** (a, b, c, …)	`departementIUT IN ('SD', 'INFO')`
		est équivalent à
		`departementIUT = 'SD' OR departementIUT = 'INFO'`

Exemple de situation

Relation (table) `etudiants`

`etudiants(INE, nom, prenom, dateNaissance, sexe, ville, groupeTD, groupeTP)`

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3
...	...	...	...	...	...	...	...

Exemple de situation

```
SELECT
 nom,
 prenom,
 groupeTP
FROM
 etudiants
WHERE
 groupeTD = 2
ORDER BY
 groupeTP, nom, prenom ;
```

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

Exemple de situation

```
SELECT
 nom,
 prenom,
 groupeTP
FROM
 etudiants
WHERE
 groupeTD = 2
ORDER BY
 groupeTP, nom, prenom ;
```

table etudiants, instance initiale

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

Exemple de situation

```
SELECT
 nom,
 prenom,
 groupeTP
FROM
 etudiants
WHERE
 groupeTD = 2
ORDER BY
 groupeTP, nom, prenom ;
```

*clause **SELECT***
indique ce que l'on souhaite obtenir comme tuple (ligne) dans le résultat de requête

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

Exemple de situation

```
SELECT
 nom,
 prenom,
 groupeTP
FROM
 etudiants
WHERE
 groupeTD = 2
ORDER BY
 groupeTP, nom, prenom ;
```

*clause **SELECT***
indique ce que l'on souhaite obtenir comme tuple (ligne) dans le résultat de requête

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

Exemple de situation

```
SELECT
 nom,
 prenom,
 groupeTP
FROM
 etudiants
WHERE
 groupeTD = 2
ORDER BY
 groupeTP, nom, prenom ;
```

*clause **WHERE***
indique quels prédicats doivent être vrais pour qu'un tuple soit dans le résultat

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1 ✖	1
2	Beh	Bob	2004-03-25	H	Metz	1 ✖	2
3	Ceh	Charlène	2004-04-28	F	Metz	1 ✖	2
4	Deh	Dexter	2004-05-12	H	Nancy	1 ✖	1
5	Euh	Edwige	2005-06-06	F	Thionville	1 ✖	1
6	Effe	Franck	2003-12-13	H	Nancy	2 ✓	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2 ✓	4
8	Hache	Hubert	2003-12-15	H	Metz	2 ✓	3
9	Ih	Isabelle	2005-10-01	F	Metz	2 ✓	4
10	Ji	Jonathan	2002-05-28	H	Metz	2 ✓	3

Exemple de situation

```
SELECT
 nom,
 prenom,
 groupeTP
FROM
 etudiants
WHERE
 groupeTD = 2
ORDER BY
 groupeTP, nom, prenom ;
```

*clause **ORDER BY***
indique dans quel ordre doivent apparaître les tuples.
ici on tri d'abord sur le groupe de TP, puis (si égalité) sur le nom et enfin sur le prénom

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
6	Effe	Franck	2003-12-13	H	Nancy	2	3
8	Hache	Hubert	2003-12-15	H	Metz	2	3
10	Ji	Jonathan	2002-05-28	H	Metz	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
9	Ih	Isabelle	2005-10-01	F	Metz	2	4

Exemple de situation

```
SELECT
 nom,
 prenom,
 groupeTP
FROM
 etudiants
WHERE
 groupeTD = 2
ORDER BY
 groupeTP, nom, prenom ;
```

Relation de départ

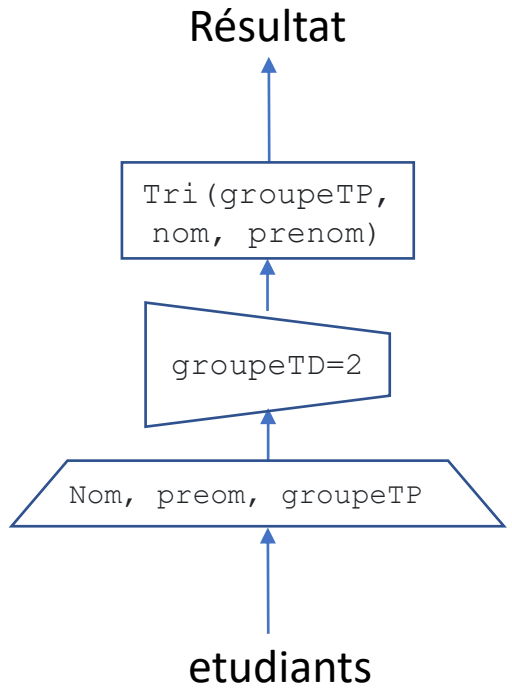
INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

Résultat

nom	prenom	groupeTP
Effe	Franck	3
Hache	Hubert	3
Ji	Jonathan	3
Geay	Géraldine	4
Ih	Isabelle	4

Exemple de situation

```
SELECT
 nom,
 prenom,
 groupeTP
FROM
 etudiants
WHERE
 groupeTD = 2
ORDER BY
 groupeTP, nom, prenom ;
```



Relation de départ

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

requête →

Résultat

nom	prenom	groupeTP
Effe	Franck	3
Hache	Hubert	3
Ji	Jonathan	3
Geay	Géraldine	4
Ih	Isabelle	4

Exemple de situation

```
SELECT
 *
FROM
 etudiants
WHERE
 dateNaissance > = '2005-01-01' ;
```

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

Exemple de situation

```
SELECT
 *
FROM
 etudiants
WHERE
 dateNaissance > = '2005-01-01' ;
```

table etudiants, instance initiale

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

Exemple de situation

```
SELECT
 *
FROM
 etudiants
WHERE
 dateNaissance > = '2005-01-01' ;
```

l'étoile (*)
indique d'utiliser tous les attributs de la relation,
dans l'ordre ou ils ont été définis.

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

Exemple de situation

```
SELECT
 *
FROM
 etudiants
WHERE
 dateNaissance > = '2005-01-01' ;
```

les valeurs littérales de dates (littéraux dates) sont écrites au format YYYY-MM-DD et entre guillemets simples (simple quotte)

Sans les guillemets, ce serait 2003 (2005 moins 1 moins 1)

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

Exemple de situation

```
SELECT
 *
FROM
 etudiants
WHERE
 dateNaissance > = '2005-01-01' ;
```

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
1	Ah	Alice	2000-01-01	F	Metz	1	1
2	Beh	Bob	2004-03-25	H	Metz	1	2
3	Ceh	Charlène	2004-04-28	F	Metz	1	2
4	Deh	Dexter	2004-05-12	H	Nancy	1	1
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
6	Effe	Franck	2003-12-13	H	Nancy	2	3
7	Geay	Géraldine	2002-07-17	F	Nancy	2	4
8	Hache	Hubert	2003-12-15	H	Metz	2	3
9	Ih	Isabelle	2005-10-01	F	Metz	2	4
10	Ji	Jonathan	2002-05-28	H	Metz	2	3

Exemple de situation

```
SELECT
 *
FROM
 etudiants
WHERE
 dateNaissance > = '2005-01-01' ;
```

Résultat de la requête

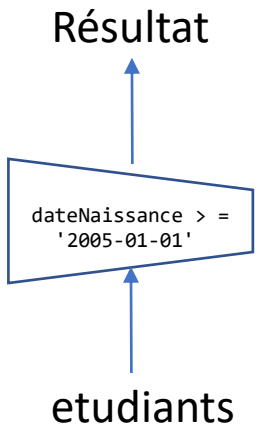
INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
9	Ih	Isabelle	2005-10-01	F	Metz	2	4

Exemple de situation

```
SELECT
 *
FROM
 etudiants
WHERE
 dateNaissance > = '2005-01-01' ;
```

Résultat de la requête

INE	nom	prenom	dateNaissance	sexe	ville	groupeTD	groupeTP
5	Euh	Edwige	2005-06-06	F	Thionville	1	1
9	Ih	Isabelle	2005-10-01	F	Metz	2	4



Exemple de situation

```
SELECT
 IF(sexe='H', 'Monsieur', 'Madame') AS civilite,
 nom,
 prenom

FROM
 etudiants

WHERE
 groupeTD = 2 AND ville = 'Metz' ;
```

Exemple de situation

"les colonnes que l'on cherche à obtenir"

```
SELECT
 IF(sexe='H', 'Monsieur', 'Madame') AS civilite,
 nom,
 prenom

FROM
  etudiants

WHERE
  groupeTD = 2 AND ville = 'Metz' ;
```

Exemple de situation

"les colonnes que l'on cherche à obtenir"

civilite	nom	prenom
...	...	...
...	...	...

```
SELECT
 IF(sexe='H', 'Monsieur', 'Madame') AS civilite,
 nom,
 prenom

FROM
  etudiants

WHERE
  groupeTD = 2 AND ville = 'Metz' ;
```

Exemple de situation

"Créer un Alias"

civilite	nom	prenom
...	...	...
...	...	...

```
SELECT
 IF(sexe='H', 'Monsieur', 'Madame') AS civilite,
 nom,
 prenom

FROM
 etudiants

WHERE
 groupeTD = 2 AND ville = 'Metz' ;
```

Exemple de situation

« Si Sexe a comme valeur ‘H’ alors ‘civilité’ prend comme valeur ‘Monsieur’, dans le cas contraire, prend la valeur ‘Madame’ "

civilite	nom	prenom
...	...	...
...	...	...

```
SELECT
 IF(sexe='H', 'Monsieur', 'Madame') AS civilite,
 nom,
 prenom

FROM
 etudiants

WHERE
 groupeTD = 2 AND ville = 'Metz' ;
```

Exemple de situation

civilite	nom	prenom
...	...	...
...	...	...

```
SELECT
 IF(sexe='H', 'Monsieur', 'Madame') AS civilite,
 nom,
 prenom

FROM
 etudiants

WHERE
 groupeTD = 2 AND ville = 'Metz' ;
```

"la source des données à utiliser"
(depuis la table `etudiants`*)*

Exemple de situation

civilite	nom	prenom
...	...	...
...	...	...

```
SELECT
 IF(sexe='H', 'Monsieur', 'Madame') AS civilite,
 nom,
 prenom

FROM
 etudiants

WHERE
 groupeTD = 2 AND ville = 'Metz' ;
```

"les conditions à vérifier pour qu'un tuple apparaisse dans le résultat"

Exemple de situation

civilite	nom	prenom
...	...	...
...	...	...

```
SELECT
 IF(sexe='H', 'Monsieur', 'Madame') AS civilite,
 nom,
 prenom

FROM
 etudiants

WHERE
 groupeTD = 2 AND ville = 'Metz' ;
```

"les conditions à vérifier pour qu'un tuple apparaisse dans le résultat"

1er prédicat

2ème prédicat

connecteur logique "ET".
Les deux prédicats doivent être vrai pour que le tuple soit sélectionné

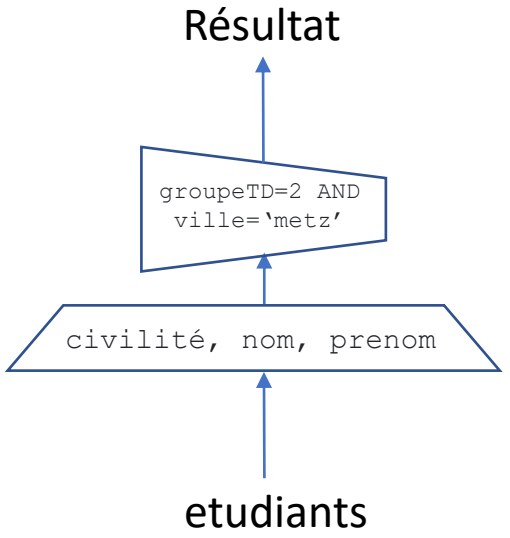
Exemple de situation

civilite	nom	prenom
…	…	…
…	…	…

```
SELECT
 IF(sexe='H', 'Monsieur', 'Madame') AS civilite,
 nom,
 prenom

FROM
 etudiants

WHERE
 groupeTD = 2 AND ville = 'Metz' ;
```



SQL et logique ternaire

Attention aux spécificités SQL qui utilise une logique ternaire :

- vrai
- faux
- inconnu

En effet, SQL possède une valeur spécifique pour l'absence de valeur (ou les valeurs manquantes) : la valeur nulle (**null**)

Exemple de prédicats :

- 5 > 2 (c'est vrai)
- 8 <= 5 (c'est faux)
- 1 = 2 (c'est faux)
- 5 > null (c'est inconnu ; ni vrai, ni faux)

SQL et logique ternaire

Attention aux spécificités de la valeur **`null`**

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

SQL et logique ternaire

Attention aux spécificités de la valeur **`null`**

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
      COUNT(*)
FROM
      employes ;
```

SQL et logique ternaire

Attention aux spécificités de la valeur **`null`**

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
      COUNT(*)
FROM
      employes ;
```

3

SQL et logique ternaire

Attention aux spécificités de la valeur **`null`**

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        COUNT(*)
FROM
        employes ;
```

3

```
SELECT
        COUNT(*)
FROM
        employes
WHERE
        salaire <= 0 OR salaire >= 0 ;
```

SQL et logique ternaire

Attention aux spécificités de la valeur **`null`**

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        COUNT(*)
FROM
        employes ;
```

3

```
SELECT
        COUNT(*)
FROM
        employes
WHERE
        salaire <= 0 OR salaire >= 0 ;
```

2

SQL et logique ternaire

Attention aux spécificités de la valeur **`null`**

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        COUNT(*)
FROM
        employes
WHERE salaire IS NULL ;
```

SQL et logique ternaire

Attention aux spécificités de la valeur **`null`**

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        COUNT(*)
FROM
        employes
WHERE salaire IS NULL ; -- IS NULL, et pas = NULL.

 -- on ne peut pas être égal à une valeur inconnue
 -- (de même pour <>, qui ne peut être utilisé)
```

SQL et logique ternaire

Attention aux spécificités de la valeur **`null`**

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        COUNT(*)
FROM
        employes
WHERE salaire IS NULL ; -- IS NULL, et pas = NULL.
```

1

```
 -- on ne peut pas être égal à une valeur inconnue
 -- (de même pour <>, qui ne peut être utilisé)
```

Autres spécificités

Une chaîne de caractère vide n'est pas une valeur nulle

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh		1700
102	Ceh	Charlène	*null*

```
SELECT
        *
FROM
        employes
WHERE
        prenom = '' ;  -- chaine vide, mais valeur connue
```

Autres spécificités

Une chaîne de caractère vide n'est pas une valeur nulle

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh		1700
102	Ceh	Charlène	*null*

```
SELECT
        *
FROM
        employes
WHERE
        prenom = '' ;  -- chaine vide, mais valeur connue
```

numEmploye	nom	prenom	salaire
101	Beh		1700

Chaines de caractères

Opérateur **`LIKE`** pour les chaines de caractères et caractères joker

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        *
FROM
        employes
WHERE nom = 'B_h' ;
```

Chaines de caractères

Opérateur **`LIKE`** pour les chaines de caractères et caractères joker

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        *
FROM
        employes
WHERE nom = 'B_h' ;
```

numEmploye	nom	prenom	salaire

Chaines de caractères

Opérateur **`LIKE`** pour les chaines de caractères et caractères joker

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        *
FROM
        employes
WHERE nom = 'B_h' ;
```

numEmploye	nom	prenom	salaire
101	Beh	Bob	1700

Chaines de caractères

Opérateur **`LIKE`** pour les chaines de caractères et caractères joker

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        *
FROM
        employes
WHERE nom LIKE 'B_h' ;
```

numEmploye	nom	prenom	salaire
101	Beh	Bob	1700

Chaines de caractères

Opérateur **`LIKE`** pour les chaines de caractères et caractères joker

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        *
FROM
        employes
WHERE nom LIKE 'B_h' ;
```

caractère joker remplaçant exactement 1 caractère

numEmploye	nom	prenom	salaire
101	Beh	Bob	1700

Chaines de caractères

Opérateur **`LIKE`** pour les chaines de caractères et caractères joker

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        *
FROM
        employes
WHERE nom LIKE '%a%' ;
```

caractère joker remplaçant 0,1 ou plusieurs caractères

Chaines de caractères

Opérateur **LIKE** pour les chaines de caractères et caractères joker

`employes(numEmploye, nom, prenom, salaire) ;`

numEmploye	nom	prenom	salaire
100	Ah	Alice	2500
101	Beh	Bob	1700
102	Ceh	Charlène	*null*

```
SELECT
        *
FROM
        employes
WHERE nom LIKE '%a%' ;
```

caractère joker remplaçant 0,1 ou plusieurs caractères

numEmploye	nom	prenom	salaire
100	Ah	**A**lice	2500
102	Ceh	Ch**a**rlène	*null*

SQL

Obtenir des données à partir de plusieurs relations
(modèle relationnel & jointures)

SELECT

les attributs, littéraux ou combinaison/calculs à sélectionner, séparés par des virgules
"les colonnes que l'on cherche à obtenir"

FROM

une relation
(***éventuellement construite à partir d'autres relations par jointure ou sous-requête***)
"la source des données à utiliser"

WHERE

Liste de prédicats, séparés par des connecteurs logiques (ET, OU, NON)
permettant de filtrer (restreindre) les tuples à obtenir.
"les conditions à vérifier pour qu'un tuple apparaisse dans le résultat"

GROUP BY

attributs à utiliser pour faire des groupes de tuples
en vu d'appliquer des fonctions d'agrégat
"les groupes de tuples à constituer pour les calculs statistiques"

HAVING

liste de prédicats, séparés par des connecteurs logiques (ET, OU, NON)
permettant de filtrer(restreindre) les tuples à obtenir depuis la relation après groupement.
"les conditions à vérifier sur les indicateurs statistiques"

ORDER BY

liste des attributs servant à ordonner (trier) les tuples obtenus.
"ce qui va servir à déterminer comment trier les tuples"

SQL

Pourquoi plusieurs tables ?

Exemple

num Commune	libelle Commune	chefLieu Departement	num Departement	libelle Departement	num Region	libelle Region	population
57001	Metz	oui	57	Moselle	44	Grand Est	
57672	Thionville	non	57	Moselle	44	Grand Est	

Pbs d'anomalies qui viennent de l'existence de dépendances fonctionnelles transitives

numCommune → libelleCommune, chefLieuDepartement, numDepartement, libelleDepartement, numRegion, libelleRegion, population

numCommune → libelleCommune, chefLieuDepartement, numDepartement, population

numDepartement → libelleDepartement, numRegion

numRegion → libelleRegion

Exemple

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, numCommuneChefLieu, #numRegion) ;

regions(numRegion, libelleRegion) ;

departements[numcommuneChefLieu] ⊆ communes[numCommune]

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	

communes

num Departement	libelle Departement	numCommune ChefLieu	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

Anomalies impossibles *:*

- *1 département (Moselle) dans 2 régions différentes (Occitanie et Grand Est).*
- *1 département (Moselle) avec 2 numéros différents (57, 58).*
- *1 département avec 2 chefs lieux*

Conséquences

Stocker les données dans plusieurs relations permet

- une meilleure qualité des données (cohérence, règles métiers)
- un accès plus rapide aux données

Conséquences

Stocker les données dans plusieurs relations permet

- une meilleure qualité des données (cohérence, règles métiers)
- un accès plus rapide aux données

…

Mais

comment présenter toutes les données nécessaires

dans une seule et même relation ?

Conséquences

AVANT

```
geographie (
  numCommune, libelleCommune, chefLieuDepartement,
  numDepartement, libelleDepartement,
  numRegion, libelleRegion, population
);
```

1 tuple de la relation `geographie` représente les informations d'une commune.

APRES

```
communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;
```

Conséquences

AVANT

```
geographie (
  numCommune, libelleCommune, chefLieuDepartement,
  numDepartement, libelleDepartement,
  numRegion, libelleRegion, population
);
```

1 tuple de la relation `geographie` représente les informations d'une commune.

APRES

```
communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;
```

"Donner la liste des départements avec leur population"

Conséquences

AVANT

```
geographie (
  numCommune, libelleCommune, chefLieuDepartement,
  numDepartement, libelleDepartement,
  numRegion, libelleRegion, population
);
```

1 tuple de la relation `geographie` représente les informations d'une commune.

```
SELECT
        libelleDepartement,
        SUM(population) AS population_Departement
FROM
        geographie
GROUP BY
        libelleDepartement ;
```

"Donner la liste des départements avec leur population"

Conséquences

```
SELECT
        libelleDepartement,
        SUM(population) AS population_Departement
FROM
        geographie
GROUP BY
        libelleDepartement ;
```

num Commune	libelle Commune	chefLieu Departement	num Departement	libelle Departement	num Region	libelle Region	population
57001	Metz	oui	57	Moselle	44	Grand Est	pop_{Metz}
57672	Thionville	non	57	Moselle	44	Grand Est	$+ pop_{Thionville}$
56260	Vannes	oui	56	Morbihan	53	Bretagne	pop_{Metz}
56121	Lorient	non	56	Morbihan	53	Bretagne	$+ pop_{Thionville}$

Conséquences

APRES

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;

```
SELECT
        libelleDepartement,
        SUM(population) AS population_Departement
FROM
        ?
GROUP BY
        libelleDepartement ;
```

"Donner la liste des départements avec leur population"

Conséquences

APRES

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	

communes

num Departement	libelle Departement	chefLieu Departement	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

Conséquences

APRES

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	

communes

num Departement	libelle Departement	chefLieu Departement	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

Conséquences

APRES

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	

communes

num Departement	libelle Departement	chefLieu Departement	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

jointure des tables `communes` et `departements` en utilisant l'attribut `numDepartement`

Conséquences

APRES

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	

communes

num Departement	libelle Departement	chefLieu Departement	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

jointure des tables `communes` et `departements` en utilisant l'attribut `numDepartement`

```
communes JOIN departements
 USING (numDepartement)
```

Conséquences

APRES

```
communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;
```

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	

communes

num Departement	libelle Departement	chefLieu Departement	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

jointure des tables `communes` et `departements` en utilisant l'attribut `numDepartement`

```
communes JOIN departements
 USING (numDepartement)
```

num Departement	num Commune	libelle Commune	population	libelle Departement	chefLieu Departement	num Region
57	57001	Metz		Moselle	57001	44
57	57672	Thionville		Moselle	57001	44
56	56260	Vannes		Morbihan	56260	53
58	58194	Nevers		Nievre	58194	27

Conséquences

APRES

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;

num Commune	libelle Commune	num Departement	population
57001	Metz	57	
57672	Thionville	57	

communes

num Departement	libelle Departement	chefLieu Departement	num Region
56	Morbihan	56260	53
57	Moselle	57001	44
58	Nievre	58194	27

departements

num Region	libelle Region
27	Bourgogne-Franche-Comté
44	Grand Est
53	Bretagne

regions

jointure des tables `communes` et `departements` en utilisant l'attribut `numDepartement`

```
communes JOIN departements
 USING (numDepartement)
```

num Departement	num Commune	libelle Commune	population	libelle Departement	chefLieu Departement	num Region
57	57001	Metz		Moselle	57001	44
57	57672	Thionville		Moselle	57001	44
56	56260	Vannes		Morbihan	56260	53
58	58194	Nevers		Nievre	58194	27

Conséquences

APRES

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;

```
SELECT
        libelleDepartement,
        SUM(population) AS population_Departement
FROM
        ?
GROUP BY
        libelleDepartement ;
```

"Donner la liste des départements avec leur population"

Conséquences

APRES

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;

```
SELECT
        libelleDepartement,
        SUM(population) AS population_Departement
FROM
         communes JOIN departements USING (numDepartement)
GROUP BY
        libelleDepartement ;
```

"Donner la liste des départements avec leur population"

Conséquences

APRES

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;

```
SELECT
        libelleDepartement,
        SUM(population) AS population_Departement
FROM
         communes JOIN departements USING (numDepartement)
GROUP BY
        libelleDepartement ;
```

"Donner la liste des régions avec leur population"

Conséquences

APRES

```
communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;
```

```
SELECT
        libelleDepartement,
        SUM(population) AS population_Departement
FROM
        ?


GROUP BY
        libelleDepartement ;
```

"Donner la liste des régions avec leur population"

Conséquences

APRES

communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;

```
SELECT
        libelleRegion,
        SUM(population) AS population_Region
FROM
         ?


GROUP BY
        libelleRegion ;
```

"Donner la liste des régions avec leur population"

Conséquences

APRES

```
communes(numCommune, libelleCommune, #numDepartement, population) ;

departements(numDepartement, libelleDepartement, communeChefLieu, #numRegion) ;

departements[communeChefLieu] ⊆ communes[numCommune]

regions(numRegion, libelleRegion) ;
```

```
SELECT
        libelleRegion,
        SUM(population) AS population_Region
FROM
        communes
        JOIN departements USING (numDepartement)
        JOIN regions USING (numRegion)
GROUP BY
        libelleRegion ;
```

"Donner la liste des régions avec leur population"

Fonctions d'agrégation

Les fonctions d'agrégation sont des fonctions idéales pour effectuer quelques statistiques de bases sur des tables.

Les principales fonctions sont les suivantes :

- SUM() pour calculer la somme sur un ensemble d'enregistrement
- AVG() pour calculer la moyenne sur un ensemble d'enregistrement
- COUNT() pour compter le nombre d'enregistrement sur une table ou une colonne distincte
- MAX() pour récupérer la valeur maximum d'une colonne sur un ensemble de ligne. Cela s'applique à la fois pour des données numériques ou alphanumérique
- MIN() pour récupérer la valeur minimum de la même manière que MAX()

Utilisation simple

L'utilisation la plus générale consiste à utiliser la syntaxe suivante :

```
SELECT fonction(colonne) FROM table
```

https://learn.microsoft.com/fr-fr/sql/

Opérateurs logiques

Les opérateurs logiques testent le caractère vrai ou faux d'une condition. Les opérateurs logiques, comme les opérateurs de comparaison, retournent un type de données **booléen** de valeur TRUE, FALSE ou UNKNOWN.

Opérateur	Signification
ALL	TRUE si tous les éléments d'un jeu de comparaisons sont TRUE.
AND	TRUE les deux expressions booléennes sont TRUE.
ANY	TRUE si n'importe quel élément d'un jeu de comparaison est TRUE.
BETWEEN	TRUE si l'opérande est situé dans une certaine plage.
EXISTS	TRUE si une sous-requête contient des lignes.
IN	TRUE si l'opérande est égal à un élément d'une liste d'expressions.
LIKE	TRUE si l'opérande correspond à un modèle.
NOT	Inverse la valeur de tout autre opérateur booléen.
OR	TRUE si l'une ou l'autre expression booléenne est TRUE.
SOME	TRUE si certains éléments d'un jeu de comparaisons sont TRUE.

https://learn.microsoft.com/fr-fr/sql/

Opérateurs arithmétiques

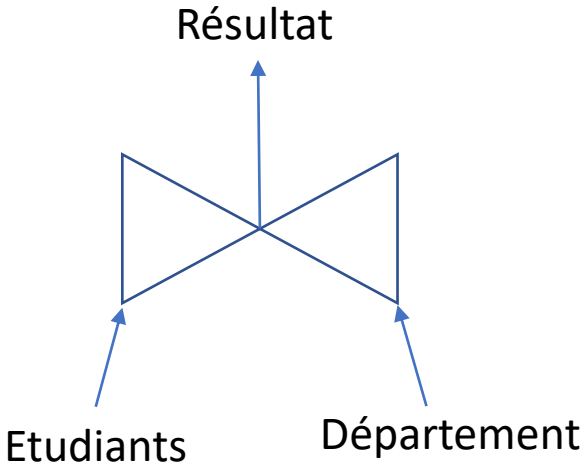
Les opérateurs arithmétiques exécutent des opérations mathématiques sur deux expressions d'un ou plusieurs types de données, à partir de la catégorie de type de données numérique.

Opérateur	Signification
+ (Ajout)	Addition
- (Soustraction)	Soustraction
* (Multiplication)	Multiplication
/ (Diviser)	Division
% (Modulo)	Retourne le reste entier d'une division. Par exemple, 12 % 5 = 2 parce que le reste de 12 divisé par 5 est 2.

https://learn.microsoft.com/fr-fr/sql/

Rappels JOIN

JOIN



etudiants

INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departements

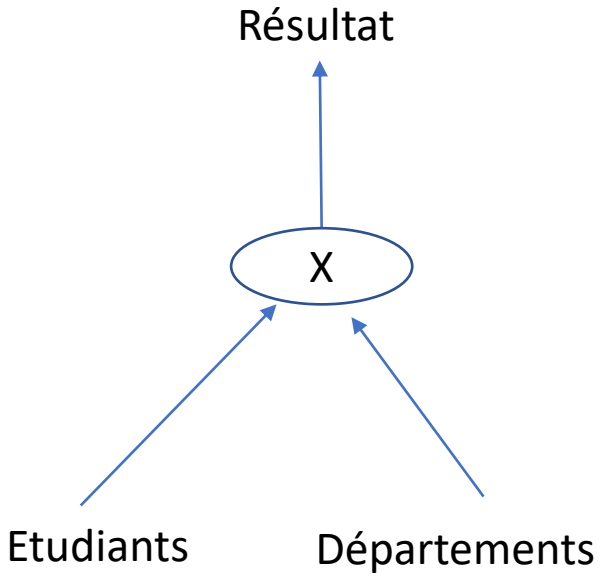
departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

```
etudiants JOIN departements
```

INE	nom	prenom	departement	departement	libelleDepartement	secteur
1	Ah	Alice	SD	SD	Science de données	tertiaire
2	Beh	Bob	INFO	SD	Science de données	tertiaire
3	Ceh	Charlène	SD	SD	Science de données	tertiaire
1	Ah	Alice	SD	INFO	Informatique	secondaire
2	Beh	Bob	INFO	INFO	Informatique	secondaire
3	Ceh	Charlène	SD	INFO	Informatique	secondaire

- sans condition, **JOIN** est équivalent à un produit cartésien (produit croisé) : il est recommandé dans ce cas d'utiliser **CROSS JOIN**, et de ne pas utiliser **JOIN** seul
- **tous les couples de tuples possibles sont présents dans le résultat**

CROSS JOIN



etudiants

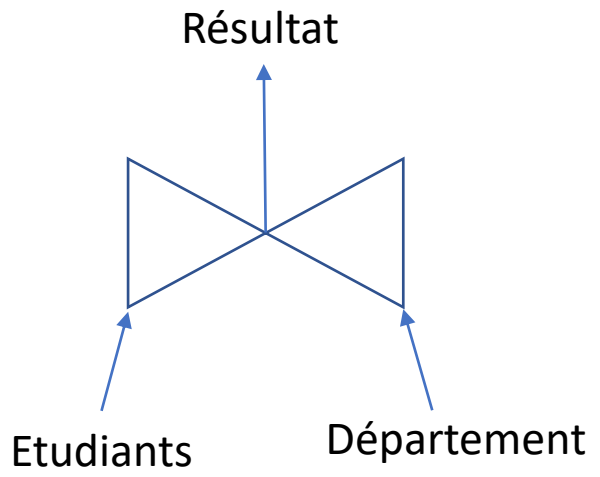
INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departements

departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

```
etudiants CROSS JOIN departements
```

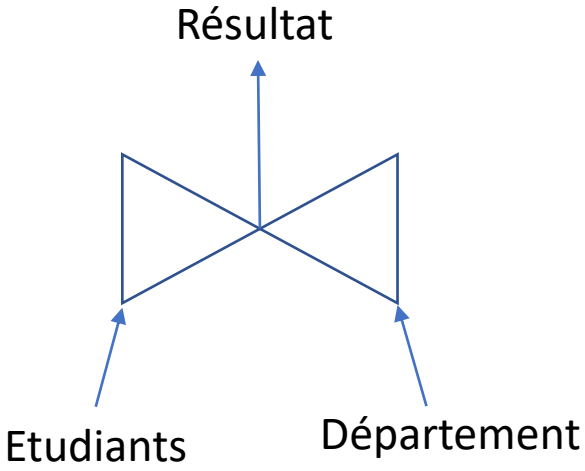
INE	nom	prenom	departement	departement	libelleDepartement	secteur
1	Ah	Alice	SD	SD	Science de données	tertiaire
2	Beh	Bob	INFO	SD	Science de données	tertiaire
3	Ceh	Charlène	SD	SD	Science de données	tertiaire
1	Ah	Alice	SD	INFO	Informatique	secondaire
2	Beh	Bob	INFO	INFO	Informatique	secondaire
3	Ceh	Charlène	SD	INFO	Informatique	secondaire



CROSS JOIN

permet d'indiquer explicitement que l'on cherche à faire un produit cartésien

NATURAL JOIN



etudiants

INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departements

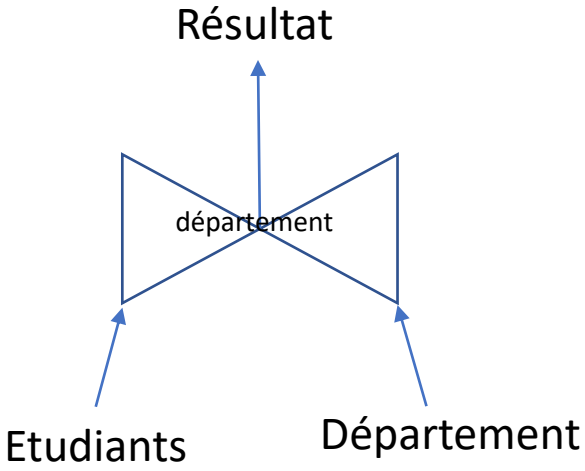
departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

```
etudiants NATURAL JOIN departements
```

departement	INE	nom	prenom	libelleDepartement	secteur
SD	1	Ah	Alice	Science de données	tertiaire
INFO	2	Beh	Bob	Informatique	secondaire
SD	3	Ceh	Charlène	Science de données	tertiaire

- On joint 2 tuples ssi ***tous*** les attributs de **même nom** dans les deux tables ont les mêmes valeurs
- **Les attributs** de jointure n'apparaissent qu'une fois, en **début de relation**
- Ici, il n y a que « département » qui est en commun, il apparait au début du résultat

JOIN …. USING



etudiants

INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departements

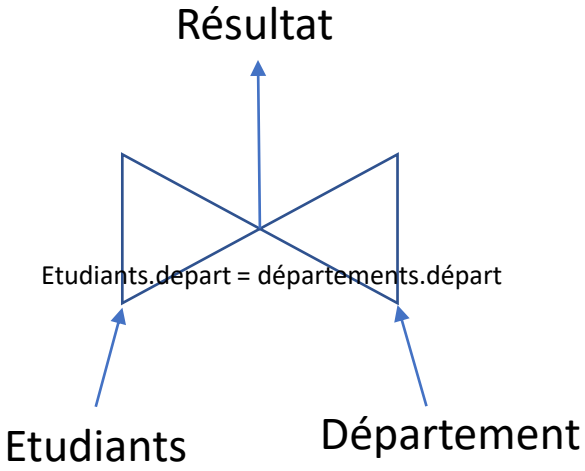
departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

```
etudiants JOIN departements USING (departement)
```

departement	INE	nom	prenom	libelleDepartement	secteur
SD	1	Ah	Alice	Science de données	tertiaire
INFO	2	Beh	Bob	Informatique	secondaire
SD	3	Ceh	Charlène	Science de données	tertiaire

- On joint 2 tuples ssi les attributs de même nom apparaissant dans **USING** ont les mêmes valeurs
- Les attributs de jointure n'apparaissent qu'une fois, **en début de relation**

JOIN .... ON



etudiants

INE	nom	prenom	departement
1	Ah	Alice	SD
2	Beh	Bob	INFO
3	Ceh	Charlène	SD

departements

departement	libelleDepartement	secteur
SD	Science de données	tertiaire
INFO	Informatique	secondaire

```
etudiants JOIN departements
ON etudiants.departement = departements.departement
```

INE	nom	prenom	departement	departement	libelleDepartement	secteur
1	Ah	Alice	SD	SD	Science de données	tertiaire
2	Beh	Bob	INFO	INFO	Informatique	secondaire
3	Ceh	Charlène	SD	SD	Science de données	tertiaire

- On joint 2 tuples ssi les conditions de la clause **ON** sont satisfaites (vrais).
- On peut utiliser autre chose que l'égalité. Les attributs peuvent avoir des noms différents.

- *Les attributs de jointure en provenance des deux tables apparaissent deux fois*

Bibliographie

- Bases de donnée, Georges Gardarin, Eyrolles, 2003
- Bases de données de la modélisation au SQL, Laurent Audibert – Collection Ellipses, 2009
- SQL, Les fondamentaux du langage (avec exercices et corrigés), Anne-Christine BISSON - Collection Ressources Informatiques, 2024.
- https://learn.microsoft.com/fr-fr/sql/ (SQL Server 2022)