

Module Introduction à la gestion des données

TD 4

Algèbre relationnelle et SQL

Hervé Owsinski
Herve.Owsinski@uphf.fr

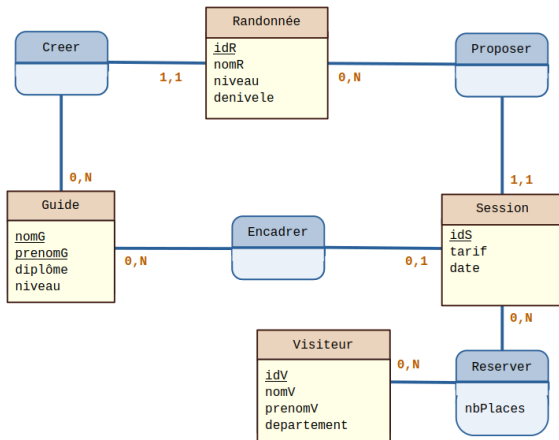
Semestre 1



2026-2027

Choix du schéma sur ce TP

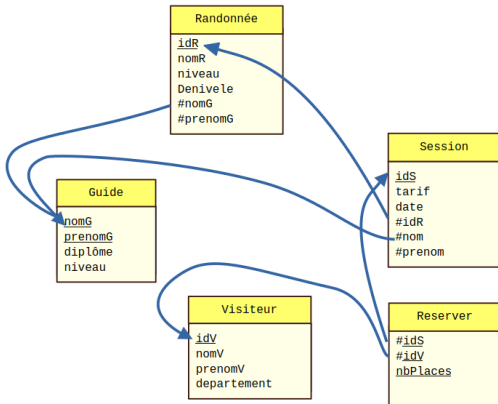
Interprétation minimaliste



Rappel

Modèle relationnel en présentation fonctionnelle

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nom, #prenom)
- **Reservation** (#idS, #idV, nbPlaces)



Cours partie 1

Projection $\pi()$
Choix des colonnes

Projection : version algèbre relationnelle

Opération de **projection** :

- Permet de choisir certains **attributs** de la relation R_e en entrée
- Notée en utilisant la fonction π (PI comme projection).

Considérons **Guide** (nomG, prenomG, diplôme, niveau)

Exemple de cours a :

Cas général :

$$\begin{aligned} R_S &= \pi_{(nomG)} (Guide) \\ &= \pi (Guide / (nomG)) \end{aligned}$$

$$\begin{aligned} R_S &= \pi_{(attributs\ voulus)} (R_e) \\ &= \pi (R_e / (attributs\ voulus)) \end{aligned}$$

Projection : version algèbre relationnelle

Opération de **projection** :

- Permet de choisir certains **attributs** de la relation R_e en entrée
- Notée en utilisant la fonction π (PI comme projection).

Considérons **Guide** (nomG, prenomG, diplôme, niveau)

Exemple de cours a :

Cas général :

$$\begin{aligned} R_S &= \pi_{(nomG)} (Guide) \\ &= \pi (Guide / (nomG)) \end{aligned}$$

$$\begin{aligned} R_S &= \pi_{(attributs\ voulus)} (R_e) \\ &= \pi (R_e / (attributs\ voulus)) \end{aligned}$$

Projection : version algèbre relationnelle

Opération de projection :

- Permet de choisir certains **attributs** de la relation R_e en entrée
- Notée en utilisant la fonction π (PI comme projection).

Considérons **Guide** (nomG, prenomG, diplôme, niveau)

Exemple de cours a :

Cas général :

$$\begin{aligned} R_S &= \pi_{(nomG)} (Guide) \\ &= \pi (Guide / (nomG)) \end{aligned}$$

$$\begin{aligned} R_s &= \pi_{(attributs\ voulus)} (R_e) \\ &= \pi (R_e / (attributs\ voulus)) \end{aligned}$$

Signification identique dans les deux cas :

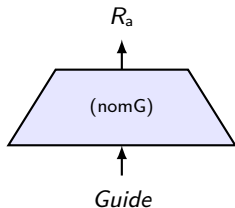
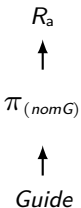
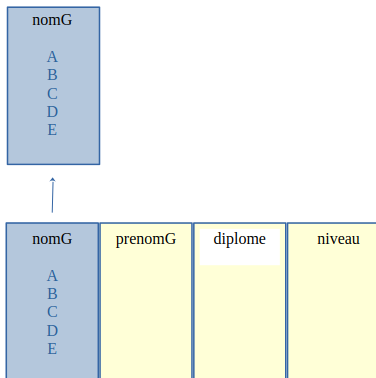
- La relation R_S est la projection des (attributs) voulus de R_e .
- La relation R_s est la projection depuis R_e des (attributs) voulus.

Projection : version arbre algébrique

Considérons **Guide** (nomG, prenomG, diplome, niveau)

$$R_a = \pi_{(nomG)} (Guide)$$

Arbre algébrique correspondant :



Projection : version SQL

$$R_a = \pi_{(nomG)} (Guide)$$

En SQL :

```
SELECT nomG  
FROM Guide;
```

L'opération de **projection** est donc liée à la clause **SELECT** de SQL.

- Derrière **SELECT** : ordre et noms des attributs à projeter
- séparés par des virgules
- Puis **FROM** suivi de la table d'entrée.

Projection : version SQL (exemple)

```
SELECT nomG  
FROM Guide;
```

nomG
Bernard
Dubois
Durand
Garnier
Leroy
Moreau
Petit
Thomas
Velais

9 rows in set

Remarque importante

Remarque importante

English Teacher :



Computer Science Student :



Projection : version algèbre relationnelle (cas B)

Considérons **Guide** (nomG, prenomG, diplôme, niveau)

Exemple de cours B :

$$R_b = \pi_{(niveau, nomG)} (Guide)$$

$$R_b = \pi (Guide /_{(niveau, nomG)})$$

Traduction ?

Projection : version algèbre relationnelle (cas B)

Considérons **Guide** (nomG, prenomG, diplôme, niveau)

Exemple de cours B :

$$R_b = \pi_{(niveau, nomG)} (Guide)$$

$$R_b = \pi (Guide /_{(niveau, nomG)})$$

Traduction ?

La relation **R_b** est la projection des tuples (niveau, nomG) de **Guide**.

Projection : version algèbre relationnelle (cas B)

Considérons **Guide** (nomG, prenomG, diplôme, niveau)

Exemple de cours B :

$$R_b = \pi_{(niveau, nomG)} (Guide)$$

$$R_b = \pi (Guide /_{(niveau, nomG)})$$

Traduction ?

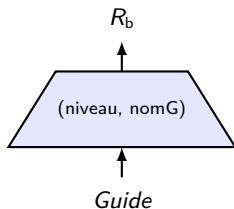
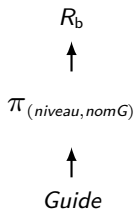
La relation **R_b** est la projection des tuples (niveau, nomG) de **Guide**.

Le schéma de sortie est donc **R_b**(niveau, nomG)

Projection : version arbre algébrique (cas B)

$$R_b = \pi_{(niveau, nomG)} (Guide) :$$

Arbre algébrique correspondant :



niveau	nomG
1	A
2	B
3	C
2	D
1	E

nomG	prenomG	diplome	niveau
A			1
B			2
C			3
D			2
E			1

Projection : version SQL (cas B)

$$R_b = \pi_{(niveau, nomG)} (Guide)$$

En SQL :

```
SELECT niveau, nomG  
FROM Guide;
```

niveau	nomG
confirmé	Bernard
débutant	Dubois
débutant	Durand
moyen	Garnier
moyen	Leroy
chevronné	Moreau
confirmé	Petit
chevronné	Thomas
chevronné	Velais

9 rows in set

Projection : version SQL (cas C)

$$R_c = \textit{Guide}$$

En SQL : (* voulant dire "tout")

```
SELECT *  
FROM Guide;
```

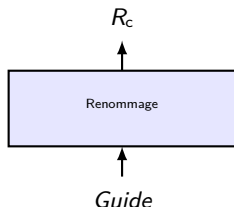
nomG	prenomG	diplome	niveau
Bernard	Claire	Guide nature	confirmé
Dubois	Antoine	BEES escalade	débutant
Durand	Marie	Guide fédéral	débutant
Garnier	Julien	BEES randonnée	moyen
Leroy	Paul	BEES randonnée	moyen
Moreau	Sophie	Guide montagne	chevronné
Petit	Luc	BEES ski	confirmé
Thomas	Arlette	Guide nature	chevronné
Velais	Denis	BEES escalade	chevronné

9 rows in set

Projection : version arbre algébrique (cas C)

$$R_c = \textit{Guide}$$

Arbre algébrique correspondant :



Pas de norme pour représenter un renommage.

Projection : gestion des doublons

$$R_b = \pi_{(niveau)} (Guide)$$

```
SELECT niveau  
FROM Guide;
```

niveau
confirmé
débutant
débutant
moyen
moyen
chevronné
confirmé

```
SELECT DISTINCT niveau  
FROM Guide;
```

niveau
confirmé
débutant
moyen
chevronné

4 rows

Projection : gestion des doublons

$$R_b = \pi_{(niveau)} (Guide)$$

- En **algèbre relationnelle**, la réponse est une **relation**, un ensemble : **doublon impossible** (le nom Dupond n'apparaîtrait qu'une seule fois).

```
SELECT niveau
FROM Guide;
```

niveau
confirmé
débutant
débutant
moyen
moyen
chevronné
confirmé

```
SELECT DISTINCT niveau
FROM Guide;
```

niveau
confirmé
débutant
moyen
chevronné

4 rows

Projection : gestion des doublons

$$R_b = \pi_{(niveau)} (Guide)$$

- En **algèbre relationnelle**, la réponse est une **relation**, un ensemble : **doublon impossible** (le nom Dupond n'apparaîtrait qu'une seule fois).
- En **SQL**, la réponse est une **table** : les **doublons sont possibles** (le nom Dupond peut apparaître plusieurs fois).

```
SELECT niveau  
FROM Guide;
```

niveau
confirmé
débutant
débutant
moyen
moyen
chevronné
confirmé

```
SELECT DISTINCT niveau  
FROM Guide;
```

niveau
confirmé
débutant
moyen
chevronné

4 rows

Projection : gestion des doublons

$$R_b = \pi_{(niveau)} (Guide)$$

- En **algèbre relationnelle**, la réponse est une **relation**, un ensemble : **doublon impossible** (le nom Dupond n'apparaîtrait qu'une seule fois).
- En **SQL**, la réponse est une **table** : les **doublons sont possibles** (le nom Dupond peut apparaître plusieurs fois).
- On peut forcer SQL à ne renvoyer qu'un exemplaire de chaque réponse avec **DISTINCT** :

```
SELECT niveau  
FROM Guide;
```

niveau
confirmé
débutant
débutant
moyen
moyen
chevronné
confirmé

```
SELECT DISTINCT niveau  
FROM Guide;
```

niveau
confirmé
débutant
moyen
chevronné

4 rows

Cours partie 2

Restriction $\sigma()$
"Choisir ses lignes"

Restriction : version algèbre relationnelle

L'opération de restriction consiste à filtrer/choisir uniquement les **tuples** issus d'une relation R_e en entrée qui respectent une *condition* fournie.

On la note en utilisant la fonction σ (sigma comme selection en anglais).

Considérons **Guide** (nomG, prenomG, diplome, niveau)

Exemple de cours D :

$$R_S = \sigma_{\text{niveau}="débutant"} (\text{Guide})$$

$$R_S = \sigma (\text{Guide} / \text{niveau}="débutant"))$$

Cas général :

$$R_S = \sigma_{\text{condition}} (R_e)$$

$$R_S = \sigma (R_e / \text{condition})$$

Restriction : version algèbre relationnelle

L'opération de restriction consiste à filtrer/choisir uniquement les **tuples** issus d'une relation R_e en entrée qui respectent une *condition* fournie.

On la note en utilisant la fonction σ (sigma comme selection en anglais).

Considérons **Guide** (nomG, prenomG, diplome, niveau)

Exemple de cours D :

$$R_S = \sigma_{\text{niveau}="débutant"} (Guide)$$

$$R_S = \sigma (Guide / \text{niveau}="débutant")$$

Cas général :

$$R_S = \sigma_{\text{condition}} (R_e)$$

$$R_S = \sigma (R_e / \text{condition})$$

Restriction : version algèbre relationnelle

L'opération de restriction consiste à filtrer/choisir uniquement les **tuples** issus d'une relation R_e en entrée qui respectent une *condition* fournie.

On la note en utilisant la fonction σ (sigma comme selection en anglais).

Considérons **Guide** (nomG, prenomG, diplome, niveau)

Exemple de cours D :

Cas général :

$$R_S = \sigma_{\text{niveau}="débutant"} (Guide)$$

$$R_S = \sigma_{condition} (R_e)$$

$$R_S = \sigma (Guide / \text{niveau}="débutant")$$

$$R_S = \sigma (R_e / condition)$$

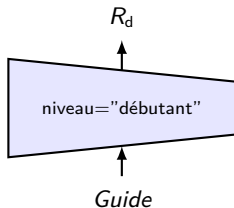
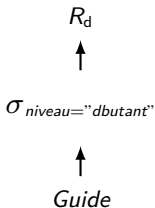
Signification identique dans les deux cas :

- La relation R_S est la restriction des lignes respectant (condition) dans R_e .
- La relation R_S est la restriction des lignes de R_e respectant (condition).

Restriction : version arbre algébrique

$$R_d = \sigma_{\text{niveau}=\text{"débutant"}} (Guide) :$$

Arbre algébrique correspondant :



nomG	prenomG	diplome	niveau
A	PA	DA	débutant
C	PC	DC	débutant
D	PD	DD	débutant

nomG	prenomG	diplome	niveau
A	PA	DA	débutant
B	PB	DB	avancé
C	PC	DC	débutant
D	PD	DD	débutant
E	PE	DE	moyen

Restriction : version SQL

$$R_d = \sigma_{\text{niveau}=\text{"débutant"}} (Guide)$$

En SQL :

```
SELECT *  
FROM Guide  
WHERE niveau = 'débutant';
```

L'opération de **restriction** est liée à la clause **WHERE** :

- Derrière **WHERE** : une expression **booléenne**.
- Connecteurs **AND OR NOT** autorisés.
- Opérateur d'égalité : **=**, comme en algo, pas comme Python !
- Opérateur de différence : **<>** (ou **!=** également)

Restriction : version SQL (exemple)

```
SELECT *  
FROM Guide  
WHERE niveau = 'débutant';
```

nomG	prenomG	diplome	niveau
Dubois	Antoine	BEES escalade	débutant
Durand	Marie	Guide fédéral	débutant

2 rows in set

Restriction : version SQL (OR et UNION)

```
SELECT *  
FROM Guide  
WHERE niveau = 'débutant' OR diplome = 'BEES escalade';
```

```
SELECT *  
FROM Guide  
WHERE niveau = 'débutant'
```

UNION

```
SELECT *  
FROM Guide  
WHERE diplome = 'BEES escalade' ;
```

Restriction : version SQL (AND et INTERSECT)

```
SELECT *  
FROM Guide  
WHERE niveau = 'débutant' AND diplome = 'BEES escalade';
```

```
SELECT *  
FROM Guide  
WHERE niveau = 'débutant'  
  
INTERSECT  
  
SELECT *  
FROM Guide  
WHERE diplome = 'BEES escalade' ;
```

Restriction : version SQL (AND et INTERSECT)

```
SELECT *  
FROM Guide  
WHERE niveau = 'débutant' AND NOT (diplome = 'BEES escalade')
```

```
SELECT *  
FROM Guide  
WHERE niveau = 'débutant'
```

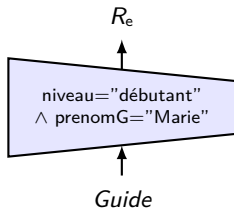
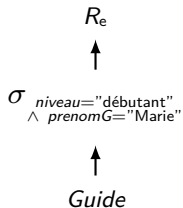
EXCEPT

```
SELECT *  
FROM Guide  
WHERE diplome = 'BEES escalade' ;
```

Restriction : version arbre algébrique (cas E)

$$R_e = \sigma \left(\text{Guide} \mid \text{niveau} = \text{"débutant"} \wedge \text{prenomG} = \text{"Marie"} \right)$$

Arbre algébrique :



nomG	prenomG	diplome	niveau
C	Marie	DC	débutant

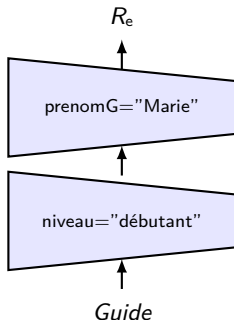
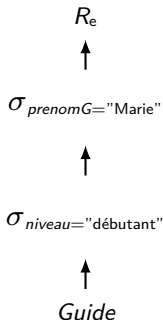
nomG	prenomG	diplome	niveau
A	PA	DA	débutant
B	PB	DB	avancé
C	Marie	DC	débutant
D	PD	DD	débutant
E	Marie	DE	moyen

Arrows indicate the selection of the row (C, Marie, DC, débutant) from the bottom table to form the top table.

Restriction : arbre algébrique interne (cas E)

$$R_e = \sigma \left(\text{Guide} / \text{niveau} = \text{"débutant"} \wedge \text{prenomG} = \text{"Marie"} \right)$$

Arbre algébrique :



nomG	prenomG	diplome	niveau
C	Marie	DC	débutant

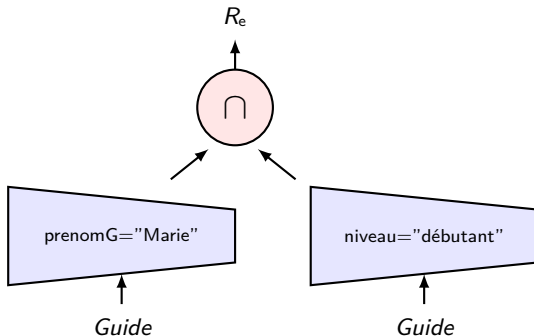
nomG	prenomG	diplome	niveau
A	PA	DA	débutant
B	PB	DB	avancé
C	Marie	DC	débutant
D	PD	DD	débutant
E	Marie	DE	moyen

Arrows indicate the mapping from the bottom table to the top table, showing the selection of the row where nomG=C, prenomG=Marie, and niveau=débutant.

Restriction : arbre algébrique interne (cas E)

$$R_e = \sigma \left(\text{Guide} / \text{niveau} = \text{"débutant"} \wedge \text{prenomG} = \text{"Marie"} \right)$$

Arbre algébrique :



Restriction : version SQL (cas E)

$R_e = \sigma \left(\text{Guide} / \text{niveau} = \text{"débutant"} \wedge \text{prenomG} = \text{"Marie"} \right) :$

SQL :

```
SELECT *  
FROM Guide  
WHERE niveau = 'débutant'  
      AND prenomG = 'Marie';
```

nomG	prenomG	diplome	niveau
Durand	Marie	Guide fédéral	débutant

1 row in set

Restriction : version SQL (cas F)

$$R_f = \sigma \left(\text{Guide} / \text{niveau} = \text{"débutant"} \vee \text{niveau} = \text{"Moyen"} \right) :$$

En SQL :

```
SELECT *  
FROM Guide  
WHERE niveau = 'débutant'  
       OR niveau = 'moyen';
```

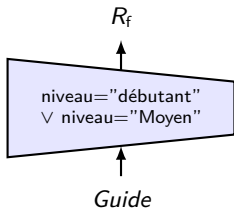
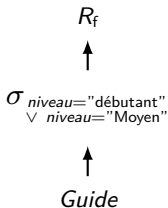
nomG	prenomG	diplome	niveau
Dubois	Antoine	BEES escalade	débutant
Durand	Marie	Guide fédéral	débutant
Garnier	Julien	BEES randonnée	moyen
Leroy	Paul	BEES randonnée	moyen

4 rows in set

Restriction : version arbre algébrique (cas F)

$$R_f = \sigma \left(\text{Guide} / \text{niveau} = \text{"débutant"} \vee \text{niveau} = \text{"Moyen"} \right) :$$

Arbre algébrique :



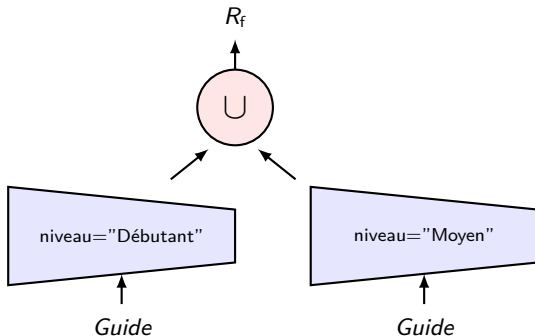
nomG	prenomG	diplome	niveau
A	PA	DA	débutant
C	PC	DC	débutant
D	PD	DD	débutant
E	PE	DE	moyen

nomG	prenomG	diplome	niveau
A	PA	DA	débutant
B	PB	DB	avancé
C	PC	DC	débutant
D	PD	DD	débutant
E	PE	DE	moyen

Restriction : version arbre algébrique (cas F)

$$R_f = \sigma \left(\text{Guide} / \text{niveau} = \text{"Débutant"} \vee \text{niveau} = \text{"Moyen"} \right) :$$

Arbre algébrique :



Projection et sélection

Composition de restriction puis projection : version SQL

$$R_g = \pi_{(nomG,diplome)}(\sigma_{(niveau="débutant" \vee niveau="moyen")}(Guide))$$

En SQL :

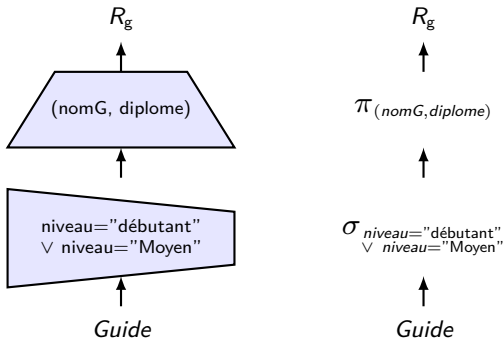
```
SELECT nomG, diplome
FROM Guide
WHERE niveau = 'débutant'
       OR niveau = 'moyen';
```

```
+-----+-----+
| nomG   | diplome           |
+-----+-----+
| Dubois | BEES escalade     |
| Durand | Guide fédéral     |
| Garnier| BEES randonnée    |
| Leroy  | BEES randonnée    |
+-----+-----+
4 rows in set
```

Composition de restriction puis projection : version arbre algébrique

$$R_g = \pi_{(nomG,diplome)}(\sigma_{(niveau="débutant" \vee niveau="moyen")}(Guide))$$

Arbre algébrique :



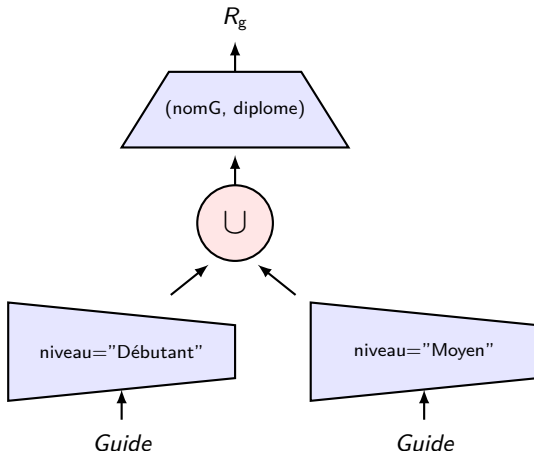
nomG	diplome
A	débutant
C	débutant
D	débutant
E	moyen

nomG	prenomG	diplome	niveau
A	PA	DA	débutant
B	PB	DB	avancé
C	PC	DC	débutant
D	PD	DD	débutant
E	PE	DE	moyen

Restriction : version arbre algébrique (cas G)

$$R_g = \pi_{(nomG,diplome)}(\sigma_{(niveau="débutant" \vee niveau="moyen")}(Guide))$$

Arbre algébrique :



Rappel : fonction composée

$$(g \circ f)(x) = g(f(x))$$

$$x \rightarrow f \rightarrow g$$

D'abord $f(x)$ puis on applique ce résultat à g .

$$(\pi \circ \sigma)(R) = \pi(\sigma(R))$$

$$R \rightarrow \sigma \rightarrow \pi$$

d'abord $\sigma(R)$ puis π .

TD SQL Questions 1 - 9

Contexte

Pour cette fiche, nous poursuivons sur le contexte de l'office de tourisme de Haute-Savoie.

Nous nous baserons sur le modèle relationnel défini dans la fiche de TD précédente.

Donnez l'arbre algébrique et la requête pour chaque question :

- 1 Donner toutes les caractéristiques des randonnées proposées.
- 2 Donner les noms et prénoms des différents guides.
- 3 Donner les noms des visiteurs du département du Nord.

Question 1

1. Donner toutes les caractéristiques des randonnées proposées.

- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)

Question 1

1. Donner toutes les caractéristiques des randonnées proposées.

- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)

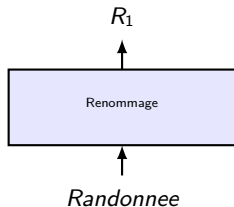
```
SELECT *  
FROM Randonnee;
```

idR	nomR	niveau	denivele	nomG	prenomG
1	La Bourgeoise	moyenne	850	Velais	Denis
2	Le Mont Veyrier	difficile	1200	Durand	Marie
3	Le Parmelan	facile	600	Petit	Luc
4	La Tournette	très difficile	1800	Moreau	Sophie
5	Le Môle	difficile	1300	Leroy	Paul
6	Le Jalouvre	très difficile	1600	Bernard	Claire

6 rows in set (0,001 sec)

Question 1

$$R_1 = (\textit{Randonnee})$$



Question 2

2. Donner les noms et prénoms des différents guides.

- **Guide** (nomG, prenomG, diplome, niveau)

Question 2

2. Donner les noms et prénoms des différents guides.

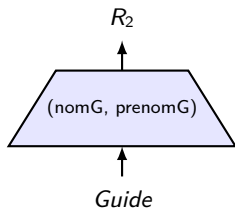
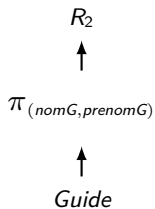
- **Guide** (nomG, prenomG, diplome, niveau)

```
SELECT nomG, prenomG
FROM Guide;
```

```
+-----+-----+
| nomG   | prenomG |
+-----+-----+
| Bernard | Claire  |
| Dubois  | Antoine |
| Durand  | Marie   |
| Garnier | Julien  |
| Leroy   | Paul    |
| Moreau  | Sophie  |
| Petit   | Luc     |
| Thomas  | Arlette |
| Velais  | Denis   |
+-----+-----+
9 rows in set (0,001 sec)
```

Question 2

$$R_2 = \pi_{(nomG, prenomG)} (Guide)$$



Question 3

3. Donner les noms des visiteurs du département du Nord.

- **Visiteur** (idV, nomV, prenomV, departement)

Question 3

3. Donner les noms des visiteurs du département du Nord.

- **Visiteur** (idV, nomV, prenomV, departement)

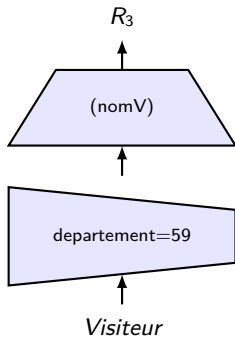
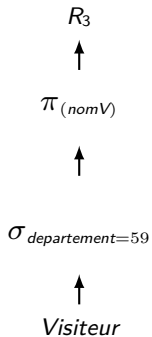
```
SELECT nomV
FROM Visiteur
WHERE departement = 59;
```

nomV
Lemoine
Petit
Girard
Gauthier
Fontaine
Leclercq
Chevalier
Roy

8 rows in set (0,000 sec)

Question 3

$$R_3 = \pi_{(nomV)} (\sigma_{departement=59} (Visiteur))$$



Question 4

4. Donner les dates des sessions de la randonnée 3.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

Question 4

4. Donner les dates des sessions de la randonnée 3.

- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)

```
SELECT date  
FROM Session  
WHERE idR = 3;
```

Empty **set**

Question 4

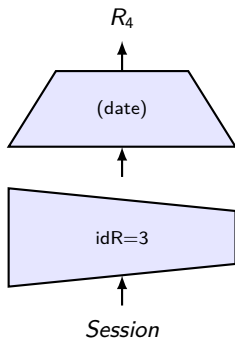
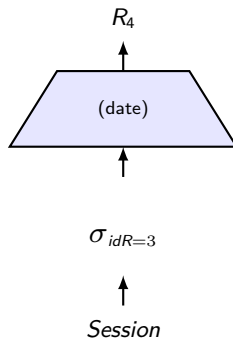
Autre exemple :

```
SELECT date
FROM Session
WHERE idR = 1;
```

```
+-----+
| date   |
+-----+
| 2025-10-01 |
| 2025-10-15 |
+-----+
2 rows in set
```

Question 4

$$R_4 = \pi_{(date)} (\sigma_{idR=3} (Session))$$



Question 5

5. Donner les noms des guides de niveau chevronné.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

Question 5

5. Donner les noms des guides de niveau chevronné.

- **Guide** (nomG, prenomG, diplome, niveau)

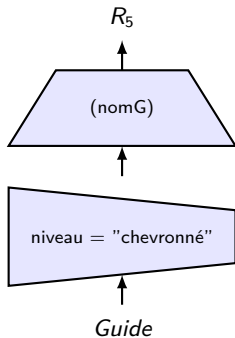
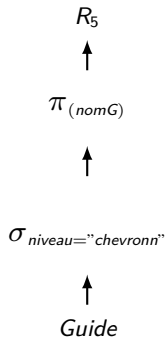
```
SELECT nomG
FROM Guide
WHERE niveau = 'chevronné';
```

```
+-----+
| nomG   |
+-----+
| Moreau |
| Thomas |
| Velais |
+-----+
```

```
3 rows in set
```

Question 5

$$R_5 = \pi_{(nomG)} (\sigma_{niveau="chevronné"} (Guide))$$



Question 6

6. Donner le nom des randonnées dont le dénivelé est supérieur à 1000 mètres.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

Question 6

6. Donner le nom des randonnées dont le dénivelé est supérieur à 1000 mètres.

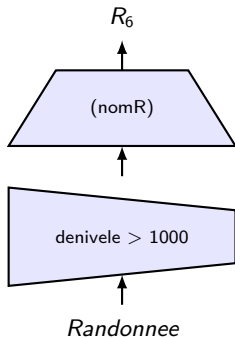
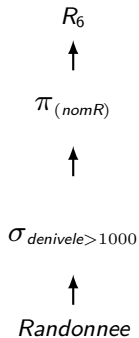
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)

```
SELECT nomR
FROM Randonnee
WHERE denivele > 1000;
```

```
+-----+
| nomR          |
+-----+
| Le Mont Veyrier |
| La Tournette   |
| Le Môle        |
| Le Jalouvre    |
+-----+
4 rows in set
```

Question 6

$$R_6 = \pi_{(nomR)} (\sigma_{denivele > 1000} (Randonnee))$$



Question 7

7. Donner toutes les informations des randonnées difficiles.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

Question 7

7. Donner toutes les informations des randonnées difficiles.

• **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)

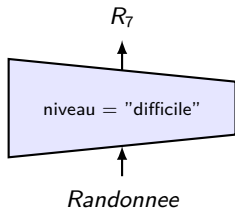
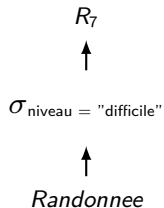
```
SELECT *  
FROM Randonnee  
WHERE niveau = 'difficile';
```

idR	nomR	niveau	denivele	nomG	prenomG
2	Le Mont Veyrier	difficile	1200	Durand	Marie
5	Le Môle	difficile	1300	Leroy	Paul

2 rows in set

Question 7

$$R_7 = \sigma_{\text{niveau}=\text{"difficile"}} (R_{\text{Randonnee}})$$



Question 8

8. Donner le nom des randonnées proposées par le guide 5.

Remarque : les guides n'ont pas d'id numérique avec notre modèle, il va falloir commencer par le trouver pour donner du sens à la question.

```
SELECT nomG, prenomG
FROM Guide;
```

nomG	prenomG	
Bernard	Claire	1
Dubois	Antoine	2
Durand	Marie	3
Garnier	Julien	4
Leroy	Paul	<---- 5
Moreau	Sophie	
Petit	Luc	
Thomas	Arlette	
Velais	Denis	

Question 8

8. Donner le nom des randonnées proposées par le guide **Paul Leroy**.

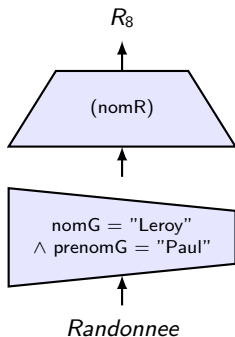
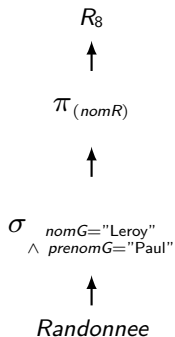
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)

```
SELECT nomR
FROM Randonnee
WHERE nomG = 'Leroy' AND prenomG = 'Paul';
```

```
+-----+
| nomR   |
+-----+
| Le Môle |
+-----+
1 row in set
```

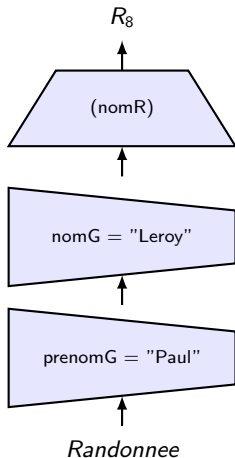
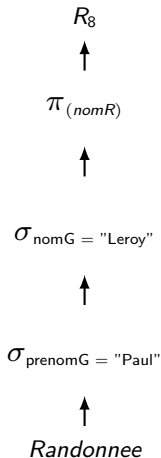
Question 8

$$R_8 = \pi_{(nomR)} (\sigma_{nomG = \text{"Leroy"} \wedge prenomG = \text{"Paul"}} (R_{donnee}))$$



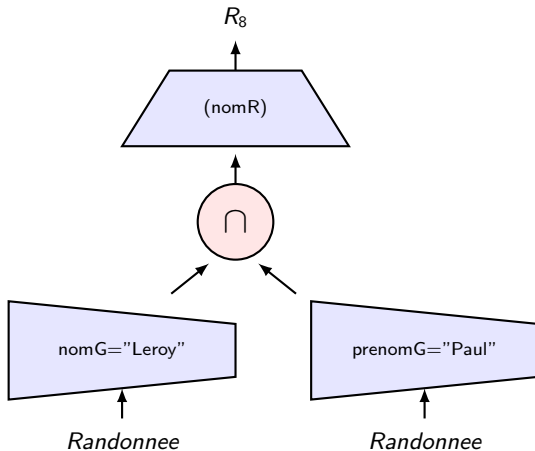
Question 8

$$R_8 = \pi_{(nomR)} (\sigma_{nomG = \text{"Leroy"}} \wedge \sigma_{prenomG = \text{"Paul"}} (R_{\text{Randonnee}}))$$



Question 8

$$R_8 = \pi_{(nomR)} (\sigma_{nomG = \text{"Leroy"} \wedge prenomG = \text{"Paul"}} (R_{donnee}))$$



Question 9

9. Donner le nom des randonnées proposées par le guide Denis Velais.

On voit bien néanmoins qu'il est très pratique que la clé primaire du guide soit (nom, prénom) : cela permet d'obtenir très rapidement son entrée sans avoir besoin de faire de jointure entre Guide et Randonnée.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

9. Donner le nom des randonnées proposées par le guide Denis Velais.

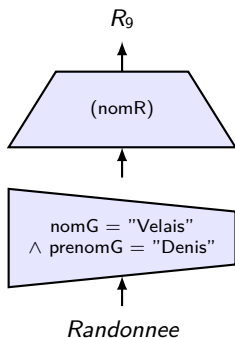
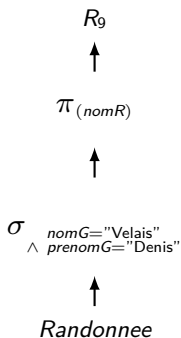
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)

```
SELECT nomR
FROM Randonnee
WHERE nomG = 'Velais' AND prenomG = 'Denis';
```

```
+-----+
| nomR          |
+-----+
| La Bourgeoise |
+-----+
1 row in set (0,000 sec)
```

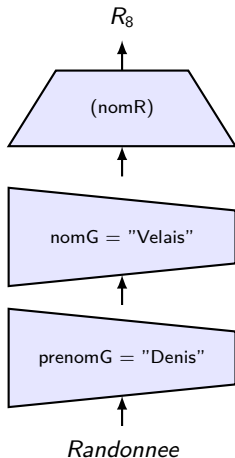
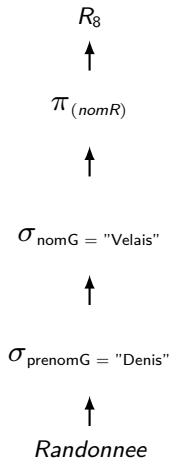
Question 9

$$R_9 = \pi_{(nomR)} (\sigma_{nomG = \text{"Vélais"} \wedge prenomG = \text{"Denis"}} (R_{donnee}))$$



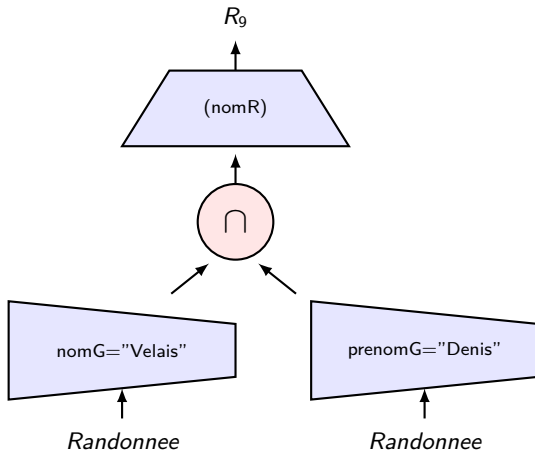
Question 9

$$R_9 = \pi_{(nomR)} (\sigma_{nomG = \text{"Velais"} \wedge prenomG = \text{"Denis"} (R_{dronnee}))$$



Question 9

$$R_9 = \pi_{(nomR)} (\sigma_{nomG = \text{"Velais"} \wedge prenomG = \text{"Denis"}} (R_{donnee}))$$



Cours partie 3

Produit cartésien $\times$

Produit cartésien de deux ensembles

L'opérateur binaire produit cartésien est symbolisé par $\times$.

Il consiste à créer un nouvel ensemble composé de couples associant chaque élément du premier ensemble à chaque élément du deuxième ensemble.

Exemple :

- si $A = \{1, 2, 3\}$;
- si $B = \{a, b\}$;
- si $C = A \times B$;
- alors C contient les couples suivants :

$$C = \{ (1, a), (1, b), (2, a), (2, b), (3, a), (3, b) \}$$

Produit cartésien de deux ensembles

L'opérateur binaire produit cartésien est symbolisé par $\times$.

Il consiste à créer un nouvel ensemble composé de couples associant chaque élément du premier ensemble à chaque élément du deuxième ensemble.

Exemple :

- si $A = \{1, 2, 3\}$;
- si $B = \{a, b\}$;
- si $C = A \times B$;
- alors C contient les couples suivants :

$$C = \{ (1, a), (1, b), (2, a), (2, b), (3, a), (3, b) \}$$

Produit cartésien de deux ensembles

L'opérateur binaire produit cartésien est symbolisé par $\times$.

Il consiste à créer un nouvel ensemble composé de couples associant chaque élément du premier ensemble à chaque élément du deuxième ensemble.

Exemple :

- si $A = \{1, 2, 3\}$;
- si $B = \{a, b\}$;
- si $C = A \times B$;
- alors C contient les couples suivants :

$$C = \{ (1, a), (1, b), (2, a), (2, b), (3, a), (3, b) \}$$

Produit cartésien de deux ensembles

L'opérateur binaire produit cartésien est symbolisé par $\times$.

Il consiste à créer un nouvel ensemble composé de couples associant chaque élément du premier ensemble à chaque élément du deuxième ensemble.

Exemple :

- si $A = \{1, 2, 3\}$;
- si $B = \{a, b\}$;
- si $C = A \times B$;
- alors C contient les couples suivants :

$$C = \{ (1, a), (1, b), (2, a), (2, b), (3, a), (3, b) \}$$

Produit cartésien de deux relations

L'opérateur binaire produit cartésien produit une nouvelle relation.

Schéma : ses **attributs** sont la concaténation de ceux des deux relations.

- si **A** (**a1**, **a2**, **a3**)
- si **B** (**b1**, **b2**, **b3**, **b4**)
- si **C** = **A** $\times$ **B**
- alors **C** (**a1**, **a2**, **a3**, **b1**, **b2**, **b3**, **b4**)

Tuple de A

+	+	+	+	+	+	+	+
	a1		a2		a3		
+	+	+	+	+	+	+	+

Tuple de C

+	+	+	+	+	+	+	+	+	+					
	a1		a2		a3		b1		b2		b3		b4	
+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Tuple de B

+	+	+	+	+	+	+	+	
	b1		b2		b3		b4	
+	+	+	+	+	+	+	+	

Produit cartésien de deux relations

Extension : les éléments de C correspondent aux tuples produits par la concaténation de chaque tuple de **A** avec chaque tuple de **B** :

Relation A

a1	a2	a3
1	2	3
4	5	6

Relation B

b1	b2	b3	b4
A	B	C	D
E	F	G	H
I	J	K	L

Relation C

a1	a2	a3	b1	b2	b3	b4
1	2	3	A	B	C	D
1	2	3	E	F	G	H
1	2	3	I	J	K	L
4	5	6	A	B	C	D
4	5	6	E	F	G	H
4	5	6	I	J	K	L

Produit cartésien de deux relations

Regardons ce nouvel exemple :

- si **A** (idA, b, #refB)
- si **B** (idB, x, y, z)
- si **C** = **A** × **B**
- alors **C** (idA, b, refB, idB, x, y, z)

Relation A

+	-----	+	-----	+	-----	+
	idA		b		refB	
+	-----	+	-----	+	-----	+

Relation C

+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+		
	idA		b		refB		idB		x		y		z	
+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+		

Relation B

+	-----	+	-----	+	-----	+	-----	+
	idB		x		y		z	
+	-----	+	-----	+	-----	+	-----	+

Pas d'ambiguïté sur les noms.

refB est une clé étrangère de **A** qui référence la clé primaire *idB* de **B**. Plus explicite de la nommer **#idB** ?

Produit cartésien de deux relations

Donnons le même nom à la clé étrangère et à la clé primaire :

- si **A** (idA, b, #idB)
- si **B** (idB, x, y, z)
- si **C** = **A** × **B**
- alors **C** (idA, b, idB, idB, x, y, z)

Relation A

+	-	-	+	-	-	+	-	-	+	-	-	+
	idA			b			idB					
+	-	-	+	-	-	+	-	-	+	-	-	+

Relation C

+	-	-	+	-	-	+	-	-	+	-	-	+	-	-	+	-	-	+
	idA		b		idB		idB		x		y		z					
+	-	-	+	-	-	+	-	-	+	-	-	+	-	-	+	-	-	+

Relation B

+	-	-	-	+	-	-	-	+	-	-	-	+	
	idB				x			y			z		
+	-	-	-	+	-	-	-	+	-	-	-	+	

AMBIGUE : C a deux attributs qui portent le même nom ! Comment lever l'ambiguïté ?

Produit cartésien de deux relations

On peut utiliser une **notation pointée** :

C (A.idA, A.b, A.idB, B.idB, B.y, B.z)

A.idB se traduit par "attribut idB de la relation A".

Si on reprend le schéma de C :

Relation C

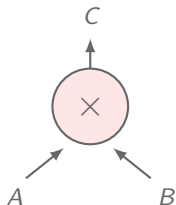
+	-----+	-----+	-----+	-----+	-----+	-----+	-----+							
	A.idA		A.b		A.idB		B.idB		B.x		B.y		B.z	
+	-----+	-----+	-----+	-----+	-----+	-----+	-----+							

Produit cartésien : Arbre algébrique

$$C = A \times B :$$

Arbre algébrique correspondant :

$$C (A.\text{id}A, A.b, A.\text{id}B, B.\text{id}B, B.x, B.y, B.z)$$



$$A (\text{id}A, b, \text{id}B)$$

$$B (\text{id}B, x, y, z)$$

Produit cartésien : SQL

$$R_g = \text{Randonnee} \times \text{Guide}$$

En SQL :

```
SELECT *  
FROM Randonnee, Guide;
```

idR	nomR	niveau	denivele	nomG	prenomG	nomG	prenomG	diplome	niveau
1	La Bourgeoise	moyenne	850	Velais	Denis	Bernard	Claire	Guide nature	confirmé
2	Le Mont Veyrier	difficile	1200	Durand	Marie	Bernard	Claire	Guide nature	confirmé
3	Le Parmelan	facile	600	Petit	Luc	Bernard	Claire	Guide nature	confirmé
4	La Tournette	très difficile	1800	Moreau	Sophie	Bernard	Claire	Guide nature	confirmé
5	Le Môle	difficile	1300	Leroy	Paul	Bernard	Claire	Guide nature	confirmé
6	Le Jalouvre	très difficile	1600	Bernard	Claire	Bernard	Claire	Guide nature	confirmé
1	La Bourgeoise	moyenne	850	Velais	Denis	Dubois	Antoine	BEES escalade	débutant
2	Le Mont Veyrier	difficile	1200	Durand	Marie	Dubois	Antoine	BEES escalade	débutant
3	Le Parmelan	facile	600	Petit	Luc	Dubois	Antoine	BEES escalade	débutant
4	La Tournette	très difficile	1800	Moreau	Sophie	Dubois	Antoine	BEES escalade	débutant
5	Le Môle	difficile	1300	Leroy	Paul	Dubois	Antoine	BEES escalade	débutant
6	Le Jalouvre	très difficile	1600	Bernard	Claire	Dubois	Antoine	BEES escalade	débutant

Produit cartésien : ambiguïté courante

Tentons ceci en SQL :

```
SELECT *  
FROM Randonnee, Guide  
WHERE nomG = 'Petit';
```

ERROR 1052 (23000): Column 'nomG' in WHERE is ambiguous

Ambigüe car on fait référence à `nomG = 'Petit'`, mais lequel ?

Remarque orthographique : [M] ambigu [F] ambiguë ou (réf. ortho.) ambigüe.

Produit cartésien : ambiguïté courante

Tentons ceci en SQL :

```
SELECT *  
FROM Randonnee, Guide  
WHERE nomG = 'Petit';
```

ERROR 1052 (23000): Column 'nomG' in WHERE is ambiguous

Ambigüe car on fait référence à `nomG = 'Petit'`, mais lequel ?

Remarque orthographique : [M] ambigu [F] ambiguë ou (réf. ortho.) ambigüe.

Produit cartésien : lever l'ambiguïté par pointage

```
SELECT *  
FROM Randonnee, Guide  
WHERE Randonnee.nomG = 'Petit';
```

On récupère les lignes du produit cartésien sur lesquelles **Randonnee.nomG** est **'Petit'**.

idR	nomR	niveau	denivele	nomG	prenomG	nomG	prenomG	diplome	niveau
3	Le Parmelan	facile	600	Petit	Luc	Bernard	Claire	Guide nature	confirmé
3	Le Parmelan	facile	600	Petit	Luc	Dubois	Antoine	BEES escalade	débutant
3	Le Parmelan	facile	600	Petit	Luc	Durand	Marie	Guide fédéral	débutant
3	Le Parmelan	facile	600	Petit	Luc	Garnier	Julien	BEES randonnée	moyen
3	Le Parmelan	facile	600	Petit	Luc	Leroy	Paul	BEES randonnée	moyen
3	Le Parmelan	facile	600	Petit	Luc	Moreau	Sophie	Guide montagne	chevronné
3	Le Parmelan	facile	600	Petit	Luc	Petit	Luc	BEES ski	confirmé
3	Le Parmelan	facile	600	Petit	Luc	Thomas	Arlette	Guide nature	chevronné
3	Le Parmelan	facile	600	Petit	Luc	Velais	Denis	BEES escalade	chevronné

9 rows in set (0,018 sec)

Par contre, il y a toujours plusieurs *nomG* dans la projection.

Produit cartésien : lever l'ambiguïté par renommage

L'opérateur de **renommage** permet de fournir **un alias (un nom alternatif)** à une table ou un attribut.

Simple changement de nom mais pratique. Quelques exemples :

- ➊ Gain de clarté : un nom est long ; il rend la requête lourde à lire ;
- ➋ On veut utiliser deux fois une table (Guide1, Guide2) ;
- ➌ Levée d'ambiguïté : deux attributs portant le même nom.

Renommage de TABLES en SQL

R est moins long à écrire que Randonnee. Idem pour G et Guide.

On peut utiliser **AS** au niveau du **FROM** :

```
SELECT nomR, R.niveau, R.nomG, R.prenomG,  
       G.nomG, G.prenomG, G.niveau  
FROM Randonnee AS R, Guide AS G  
WHERE G.nomG = 'Petit';
```

nomR	niveau	nomG	prenomG	nomG	prenomG	niveau
La Bourgeoise	moyenne	Velais	Denis	Petit	Luc	confirmé
Le Mont Veyrier	difficile	Durand	Marie	Petit	Luc	confirmé
Le Parmelan	facile	Petit	Luc	Petit	Luc	confirmé
La Tournette	très difficile	Moreau	Sophie	Petit	Luc	confirmé
Le Môle	difficile	Leroy	Paul	Petit	Luc	confirmé
Le Jalouvre	très difficile	Bernard	Claire	Petit	Luc	confirmé

Reste une ambiguïté au niveau des noms d'attributs projetés.

Renommage d'ATTRIBUTS en SQL

Reste donc à modifier les noms des attributs projetés.

On peut utiliser **AS** au niveau du **SELECT** :

```
SELECT nomR, R.niveau AS difficulté,  
        R.nomG AS nCreateur, R.prenomG AS pCreateur,  
        G.nomG AS nGuide, G.prenomG AS pGuide,  
        G.niveau AS niveau  
FROM Randonnee AS R, Guide AS G  
WHERE G.nomG = 'Petit';
```

nomR	difficulté	nCreateur	pCreateur	nGuide	pGuide	niveau
La Bourgeoise	moyenne	Velais	Denis	Petit	Luc	confirmé
Le Mont Veyrier	difficile	Durand	Marie	Petit	Luc	confirmé
Le Parmelan	facile	Petit	Luc	Petit	Luc	confirmé
La Tournette	très difficile	Moreau	Sophie	Petit	Luc	confirmé
Le Môle	difficile	Leroy	Paul	Petit	Luc	confirmé
Le Jalouvre	très difficile	Bernard	Claire	Petit	Luc	confirmé

Renommage en SQL (sans AS)

On notera que la plupart des implémentations SQL permettent de se passer du AS, mais est-ce vraiment clair ?

```
SELECT nomR,  
       R.niveau difficile,  
       R.nomG nCreateur,  
       R.prenomG pCreateur,  
       G.nomG nGuide,  
       G.prenomG pGuide,  
       G.niveau niveauG  
FROM Randonnee R, Guide G  
WHERE G.nomG = "Petit";
```

Renommage en SQL (exemple)

Exemple d'utilisation avec une table ayant deux noms :

Récupérer tous les couples possibles de Guide de façon à organiser des binômes de travail (un confirmé, un débutant) ?

```
SELECT G1.nomG AS tuteur, G2.nomG AS stagiaire
FROM Guide G1, Guide G2
WHERE G1.niveau = 'Confirmé'
      AND G2.niveau = 'Débutant';
```

tuteur	stagiaire
Bernard	Dubois
Petit	Dubois
Bernard	Durand
Petit	Durand

Renommage en Algèbre Relationnelle

L'opérateur renommage se note ρ (lettre RHO, comme Rename).

Renommage d'attributs : on veut **Randonnee** où les noms/prenoms du guide créateur deviennent

de *nomG* *prenomG* à *nCreateur*, *pCreateur*.

Exemple de cours H :

$$R_h = \rho_{nCreateur \leftarrow nomG, pCreateur \leftarrow prenomG} (Randonnee)$$

$$R_h = \rho (Randonnee / nCreateur \leftarrow nomG, pCreateur \leftarrow prenomG)$$

La relation R_h possède le même schéma de tuples que *Randonnee* à part le changement de noms des deux attributs.

Renommage en Algèbre Relationnelle

L'opérateur renommage se note ρ (lettre RHO, comme Rename).

Renommage de relations : créons une nouvelle relation **R**, copie exacte de **Randonnee**

Exemple de cours I :

$$R_i = \rho_R (Randonnee)$$

$$R = Randonnee$$

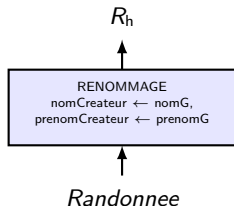
La deuxième forme est utilisable si le renommage est la dernière fonction. Par contre, si le renommage est une étape intermédiaire, la première forme est indispensable.

Renommage : Arbre Algébrique

Pas de schématisation normalisée officielle.

$$R_h = \rho_{nCreateur \leftarrow nomG, pCreateur \leftarrow prenomG} (R_{donnee})$$

Arbre algébrique correspondant :

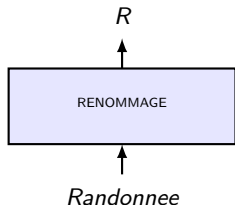


Renommage : Arbre Algébrique

Pas de schématisation normalisée officielle.

$$R = \text{Randonnee}$$

Arbre algébrique correspondant :



Cours partie 4

Jointure

Jointure : principe

Faire une **jointure** consiste à construire une relation à partir des tuples de plusieurs autres relations.

Le **produit cartésien** réalise **toutes les associations**.

Une **jointure** ne prend en compte que les **liaisons pertinentes**.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Reservation** (#idS, #idV, nbPlaces)

Ce schéma relationnel fait le lien entre une **Reservation** et un **Visiteur** à l'aide des attributs

`Reservation.idV` et `Visiteur.idV`.

Jointure : principe

Faire une **jointure** consiste à construire une relation à partir des tuples de plusieurs autres relations.

Le **produit cartésien** réalise **toutes les associations**.

Une **jointure** ne prend en compte que les **liaisons pertinentes**.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Reservation** (#idS, #idV, nbPlaces)

Ce schéma relationnel fait le lien entre une **Reservation** et un **Visiteur** à l'aide des attributs
Reservation.idV et Visiteur.idV.

Jointure : principe

Voici le produit cartésien $Reservation \times Visiteur$:

```
SELECT * FROM Reservation Re, Visiteur V;
```

Re.idS	Re.idV	Re.nbPlaces	V.idV	V.nomV	V.prenomV	V.departement
1	[1]	2	[1]	Martin	Alice	75
1	2	1	1	Martin	Alice	75
1	1	2	2	Lemoine	Paul	59
1	[2]	1	[2]	Lemoine	Paul	59
1	1	2	3	Nguyen	Thi	69
1	2	1	3	Nguyen	Thi	69
2	[3]	3	[3]	Nguyen	Thi	69
1	1	2	4	Dubois	Marc	75

Il y a parfois correspondance entre Re.idV et V.idV :

Le visiteur 1 a vraiment réservé 2 places pour la session 1.

Le visiteur 2 a vraiment réservé 1 place pour la session 1.

Le visiteur 3 a vraiment réservé 3 places pour la session 2.

Le reste n'est pas une association fictive de deux tuples.

Jointure : principe

Regardons ce que donne le produit cartésien $Reservation \times Visiteur$:

```
SELECT * FROM Reservation Re, Visiteur V;
```

Re.idS	Re.idV	Re.nbPlaces	V.idV	V.nomV	V.prenomV	V.departement
1	[1]	2	[1]	Martin	Alice	75
1	2	1	1	Martin	Alice	75
1	1	2	2	Lemoine	Paul	59
1	[2]	1	[2]	Lemoine	Paul	59
1	1	2	3	Nguyen	Thi	69
1	2	1	3	Nguyen	Thi	69
2	[3]	3	[3]	Nguyen	Thi	69
1	1	2	4	Dubois	Marc	75

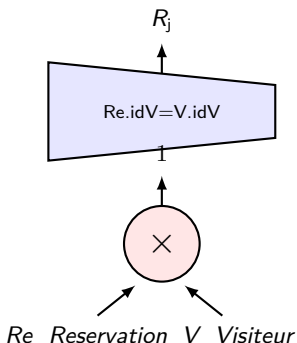
Réaliser la **jointure** se fait en deux temps :

- faire le produit cartésien $R \times V$
- se restreindre aux lignes sur lesquelles $Re.idV = V.idV$

Jointure (1 attribut) : version arbre algébrique

$$R_j = \sigma_{Re.idV = V.idV} (Re \times V)$$

Arbre algébrique :



#idS	#idV	nbP	idV	nomV	z
1	1	7	1	Alice	11
1	1	4	2	Bob	22
1	1	2	3	Claire	33
2	1	7	1	Alice	11
2	1	4	2	Bob	22
2	1	2	3	Claire	33
3	2	7	1	Alice	11
3	2	4	2	Bob	22
3	3	2	3	Claire	33

#idS	#idV	nbP
1	1	7
2	1	4
3	3	2

idV	nomV	dptV
1	Alice	11
2	Bob	22
3	Claire	33

Jointure (1 attribut) : version SQL

$$R_j = \sigma_{Re.idV = V.idV} (Re \times V)$$

Traduction en français : "Place derrière chaque tuple de S le tuple d'un visiteur qui a fait une réservation pour cette session". Le numéro du visiteur est donc le même pour le tuple de Visiteur et le tuple de Reservation.

En SQL :

```
SELECT *  
FROM Reservation Re, Visiteur V  
WHERE Re.idV = V.idV;
```

Jointure (1 attribut) : version SQL

```
SELECT *  
FROM Reservation Re, Visiteur V  
WHERE Re.idV = V.idV;
```

idS	idV	nbPlaces	idV	nomV	prenomV	departement
1	1	2	1	Martin	Alice	75
1	2	1	2	Lemoine	Paul	59
2	3	3	3	Nguyen	Thi	69
2	4	2	4	Dubois	Marc	75
3	5	1	5	Petit	Anne	59
3	6	2	6	Rousseau	Lucie	33
4	7	2	7	Morel	Jean	13
5	8	1	8	Faure	Claire	75
5	9	3	9	Girard	Pierre	59
6	10	2	10	Michel	Sophie	33

Jointure $\bowtie$: Algèbre relationnelle

La jointure tellement courante qu'un opérateur spécifique noté $A \bowtie_{cond} B$ a été rajouté.

A B sont des relations, *cond* la condition de jointure.

Trois écritures équivalentes :

$$R_j = \sigma_{A.cle=B.cle} (A \times B)$$

$$R_j = \bowtie_{A.cle=B.cle} (A, B)$$

$$R_j = A \bowtie_{A.cle=B.cle} B$$

$$R_j = \rho_{Re}(Randonnee) \bowtie_{Re.idV=V.idV} \rho_V(Visiteur)$$

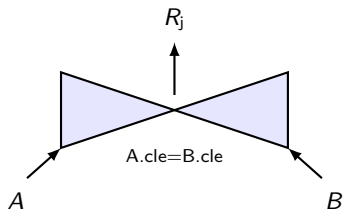
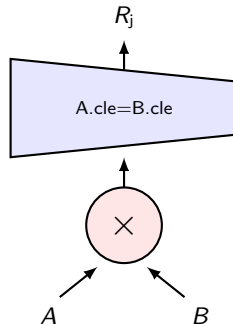
$$R_j = Re \bowtie_{Re.idV=V.idV} V$$

Remarque : opérateur JOINTURE pas strictement nécessaire : c'est composition d'un produit cartésien et d'une restriction.

Jointure : 2 Arbres algébriques !

$$R_j = \sigma_{A.cle=B.cle} (A \times B)$$

$$R_j = A \bowtie_{A.cle=B.cle} B$$



Jointure (1 attribut) : 2 version en SQL aussi

```
SELECT *  
FROM Reservation Re  
JOIN Visiteur V ON Re.idV = V.idV;
```

idS	idV	nbPlaces	idV	nomV	prenomV	departement
1	1	2	1	Martin	Alice	75
1	2	1	2	Lemoine	Paul	59
2	3	3	3	Nguyen	Thi	69
2	4	2	4	Dubois	Marc	75
3	5	1	5	Petit	Anne	59
3	6	2	6	Rousseau	Lucie	33
4	7	2	7	Morel	Jean	13
5	8	1	8	Faure	Claire	75
5	9	3	9	Girard	Pierre	59
6	10	2	10	Michel	Sophie	33

Jointure (1 attribut) : 2 SQL et plus...

L'ordre de cette jointure importe peu :

```
SELECT *  
FROM Reservation Re  
JOIN Visiteur      V ON Re.idV = V.idV;
```

```
SELECT *  
FROM Reservation Re  
JOIN Visiteur      V ON V.idV = Re.idV;
```

```
SELECT *  
FROM Visiteur      V  
JOIN Reservation Re ON V.idV = Re.idV;
```

Jointure (2 attributs) : Algèbre relationnelle

Si les clés sont des couples, il y a 2 conditions à noter : c'est le cas entre **Guide** et **Randonnee**.

- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)

$$R_j = \sigma_{R.nomG=G.nomG \wedge R.prenomG=G.prenomG} (G \times R)$$

$$R_j = G \bowtie_{R.nomG=G.nomG \wedge R.prenomG=G.prenomG} R$$

Jointure (2 attributs) : SQL

En SQL, on a aussi deux façons de formuler la demande :

```
SELECT *  
FROM Randonnee R, Guide G  
WHERE R.nomG = G.nomG AND R.prenomG = G.prenomG;
```

```
SELECT *  
FROM Randonnee R  
JOIN Guide      G ON R.nomG = G.nomG  
                  AND R.prenomG = G.prenomG;
```

Jointure (2 attributs) : SQL v1

```
SELECT *  
FROM Randonnee R, Guide G  
WHERE R.nomG = G.nomG AND R.prenomG = G.prenomG;
```

idR	nomR	niveau	den.	nomG	prenomG	nomG	prenomG	diplome	niveau
1	La Bourgeoise	moyenne	850	Velais	Denis	Velais	Denis	BEES escalade	chevronné
2	Le Mont Veyrie	difficile	1200	Durand	Marie	Durand	Marie	Guide fédéral	débutant
3	Le Parmelan	facile	600	Petit	Luc	Petit	Luc	BEES ski	confirmé
4	La Tournette	très difficile	1800	Moreau	Sophie	Moreau	Sophie	Guide montagne	chevronné
5	Le Môle	difficile	1300	Leroy	Paul	Leroy	Paul	BEES randonnée	moyen
6	Le Jalouvre	très difficile	1600	Bernard	Claire	Bernard	Claire	Guide nature	confirmé

Jointure (2 attributs) : SQL v2

```
SELECT *  
FROM Randonnee R  
JOIN Guide G ON R.nomG = G.nomG AND R.prenomG = G.prenomG;
```

idR	nomR	niveau	denivele	nomG	prenomG	nomG	prenomG	diplome	niveau
1	La Bourgeoise	moyenne	850	Velais	Denis	Velais	Denis	BEES escalade	chevronné
2	Le Mont Veyrier	difficile	1200	Durand	Marie	Durand	Marie	Guide fédéral	débutant
3	Le Parmelan	facile	600	Petit	Luc	Petit	Luc	BEES ski	confirmé
4	La Tournette	très difficile	1800	Moreau	Sophie	Moreau	Sophie	Guide montagne	chevronné
5	Le Môle	difficile	1300	Leroy	Paul	Leroy	Paul	BEES randonnée	moyen
6	Le Jalouvre	très difficile	1600	Bernard	Claire	Bernard	Claire	Guide nature	confirmé

JOIN permet de séparer les conditions en deux groupes :

- Derrière le **ON** : condition de jointure ;
- Derrière le **WHERE** : restrictions générales.

Jointure et restriction : SQL

```
SELECT *  
FROM Randonnee R  
JOIN Guide G ON R.nomG = G.nomG AND R.prenomG = G.prenomG  
WHERE G.niveau = 'Confirmé' OR G.diplome = 'Guide montage';
```

vs

```
SELECT *  
FROM Randonnee R, Guide G  
WHERE R.nomG = G.nomG  
      AND R.prenomG = G.prenomG  
      AND (G.niveau = 'Confirmé' OR G.diplome = 'Guide montage');
```

Jointure (2 attributs) : cas complexe

Cherchons les noms des visiteurs qui ont réservé au moins une fois 10 places lors d'une visite :

```
SELECT V.nomV
FROM Visiteur V
JOIN Reservation Re ON V.idV = Re.idV
WHERE Re.nbPlaces >= 10;
```

Si pas d'ambiguïté de noms, on peut se passer de la notation pointée sur les noms uniques.

```
SELECT nomV
FROM Visiteur V
JOIN Reservation Re ON V.idV = Re.idV
WHERE nbPlaces >= 10;
```

Jointure : 2 Arbres algébriques

