SQL et les bases de données relationnelles

SQL fonctionnel : l'algèbre relationnelle

Guillaume Raschia — Nantes Université

originaux de Philippe Rigaux, CNAM

Dernière mise-à-jour : 7 décembre 2023

Plan de la session

SQL, langage algébrique (S4.1)

La jointure (S4.2)

Expressions algébriques (S4.3)

SQL, langage algébrique (S4.1)

Algèbre relationnelle : les opérateurs

SQL propose un autre type d'interrogation, fonctionnelle, basée sur l'algèbre relationnelle.

L'algèbre est un ensemble de 6 opérateurs, qui présentent deux propriétés essentielles

- Clôture : un opérateur s'applique à des relations et produit une relation
- Composition : un opérateur peut prendre en entrée le résultat d'un autre pour définir des requêtes algébriques complexes.

Cette section présente les 6 opérateurs nécessaires et suffisants.

La projection, π_X

$\pi_{A_1,A_2,\ldots,A_k}(R)$ construit une relation contenant tous les nuplets de R, dans lesquels seuls les attributs $A_1, A_2, \ldots A_k$ sont conservés.

Exemple

nom et lieu des logements : $\pi_{\texttt{nom,lieu}}(\texttt{Logement})$

```
select nom, lieu
from   Logement
```

nom	lieu
Causses	Cévennes
Génépi	Alpes
U Pinzutu	Corse
Tabriz	Bretagne

`select *` permet de conserver tous les attributs.

La sélection, σ_F

La sélection $\sigma_F(R)$ s'applique à une relation, R, et en extrait les nuplets qui satisfont F.

$$\sigma_{\texttt{lieu='Corse'}}(\texttt{Logement})$$

En SQL :

```sql
select *
from   Logement
where  lieu = 'Corse'
```

Les comparaisons s'écrivent $A\ \Theta\ B$ ou $A\ \Theta\ a$, avec $\Theta \in \{=, <, >, \leq, \geq\}$.

Le produit cartésien, $\times$

$R \times S$ produit une relation dans laquelle chaque nuplet de R est associé à chaque nuplet de S.

$R =$

A	B
a	b
x	y

$S =$

C	D
c	d
u	v
x	y

Et voici le résultat du produit cartésien :

$R \times S =$

A	B	C	D
a	b	c	d
a	b	u	v
a	b	x	y
x	y	c	d
x	y	u	v
x	y	x	y

En SQL, `cross join`

En SQL, le produit cartésien s'exprime avec `cross join`. On place l'expression algébrique dans la clause `from`.

```
select * from R cross join S
```

`R cross join S` définit une nouvelle relation, les autres opérateurs (sélection, projection) s'appliquent à cette relation.

Le `from` définit donc une relation calculée.

La version déclarative $\{(r, s) \mid R(r) \wedge S(s)\}$ s'écrit en SQL :

```
select r.*, s.* from R as r, S as s;   -- simplifiée en:
select * from R, S
```

Problème des noms d'attribut ambigus

Il peut arriver que les deux relations aient des attributs qui ont le même nom.

Il est alors préférable – nécessaire – de donner un nom distinct à chaque attribut.

Relation R =

A	B
a	b
x	y

Relation T =

A	B
m	n
o	p

$R \times T$ a pour schéma $(R.A, R.B, T.A, T.B)$ et présente donc des ambiguités.

Le renommage ρ

L'expression $\rho_{A \to C, B \to D}(T)$ renomme A en C et B en D dans la relation T.

$\rho_{A \to C, B \to D}(T) =$

C	D
m	n
o	p

Le résultat du produit cartésien s'écrit alors :

$R \times \rho_{A \to C, B \to D}(T) =$

A	B	C	D
a	b	m	n
a	b	o	p
x	y	m	n
x	y	o	p

Renommage en SQL : as

La requête

```
select Activité.codeLogement,
       Séjour.codeLogement
from   Activité cross join Séjour
```

peut engendrer une erreur *duplicate field – ou column – name*

Correction :

```
select Activité.codeLogement as codeL1,
       Séjour.codeLogement as codeL2
from   Activité cross join Séjour
```

Le `as` permet aussi de renommer des relations (désigner une variable nuplet).

L'union, $\cup$

$R \cup S$ produit une relation contenant l'union de R et de S (qui doivent avoir le même schéma).

$R =$

A	B
a	b
x	y

$S =$

C	D
c	d
u	v
x	y

Et voici le résultat de l'union :

$R \times S = R \cup S =$

A	B
a	b
c	d
u	v
x	y

Union en SQL : `union`

La requête donnant l'union des noms de lieu et des noms de logement

```
select lieu from Logement
  union
select région as lieu from Voyageur
```

Nb : le schéma du résultat est `(lieu)`

- Rarement utilisée, mais indispensable (pas d'autre expression possible) en cas de besoin
- Avantageusement simulée par le connecteur logique `or` dans la clause `where`.

La différence, –

$R - S$ produit une relation contenant les nuplets de R qui ne sont pas dans S.

R et S doivent avoir le même schéma.

$R =$

A	B
a	b
x	y

$S =$

A	B
c	d
u	v
x	y

Relation $R - S =$

A	B
a	b

Différence en SQL : `except`

Les lieux de villégiature desquels aucun voyageur n'est originaire.

```
select lieu from Logement
  except
select région as lieu from Voyageur
```

- Très peu pratique à cause de la contrainte sur les schémas
- La version déclarative, `not exists`, est plus intuitive
- Parfois même, une simple négation, `not`, dans la clause `where` suffit.

À retenir

- Syntaxe et signification des 6 opérateurs
- Clôture : toute opération s'applique à des relations et produit une relation
- Composition : on crée des requêtes complexes en combinant des opérateurs, pas en créant des opérateurs super-complexes
- En SQL : l'algèbre opère sur des relations (ensembles finis de nuplets), alors que dans la perspective déclarative, SQL exprime des conditions sur un – ou plusieurs – nuplet(s).

Principales nouveautés pour SQL : `cross join`, `union`, `except`

La jointure (S4.2)

Algèbre relationnelle : la jointure algébrique

La jointure $R \underset{R.A=S.B}{\bowtie} S$ est la composition du produit cartésien et de la sélection.

$R \underset{R.A=S.B}{\bowtie} S$ est équivalent à $\sigma_{R.A=S.B}(R \times S)$.

Opération très importante.

- Elle permet de créer une relation
  - associant des nuplets distincts
  - sous condition
- C'est aussi une opération potentiellement coûteuse.

Cette section présente la jointure algébrique et sa syntaxe SQL.

Reprenons le produit cartésien `Logement` × `Activité`

code	nom	capacité	type	lieu	codeLogement	codeActivité
ca	Causses	45	Auberge	Cévennes	ca	Randonnée
ge	Génépi	134	Hôtel	Alpes	ca	Randonnée
pi	U Pinzutu	10	Gîte	Corse	ca	Randonnée
ta	Tabriz	34	Hôtel	Bretagne	ca	Randonnée
ca	Causses	45	Auberge	Cévennes	ge	Piscine
ge	Génépi	134	Hôtel	Alpes	ge	Piscine
pi	U Pinzutu	10	Gîte	Corse	ge	Piscine
ta	Tabriz	34	Hôtel	Bretagne	ge	Piscine
...	...	...	...	...	...	...

Beaucoup de lignes (probablement) sans intérêt.

Jointure : Logement $\bowtie_{\text{code=codeLogement}}$ Activité

On conserve les nuplets avec le même code logement seulement.

code	nom	capacité	type	lieu	codeLogement	codeActivité
ca	Causses	45	Auberge	Cévennes	ca	Randonnée
ge	Génépi	134	Hôtel	Alpes	ge	Piscine
ge	Génépi	134	Hôtel	Alpes	ge	Ski
pi	U Pinzutu	10	Gîte	Corse	pi	Plongée
pi	U Pinzutu	10	Gîte	Corse	pi	Voile

On a créé une nouvelle table en combinant des nuplets liés par une condition.

Il s'agit d'un sous-ensemble du produit cartésien des deux relations.

En SQL : `join ... on ...`

La jointure algébrique s'effectue en SQL dans la clause `from`.

```
select *
from Logement join Activité on (code = codeLogement)
```

Rappel : la syntaxe déclarative est la suivante :

$$\{(\ell, a) \mid \texttt{Logement}(\ell) \wedge \texttt{Activité}(a) \wedge \ell.\texttt{code} = a.\texttt{codeLogement}\}$$

```
select *
from   Logement as l, Activité as a
where  l.code = a.codeLogement
```

Interprétations différentes, résultat identique.

Résolution des ambiguités

La requête suivante renvoie une erreur à cause de l'ambiguité sur `codeLogement`.

```
select *
from Activité join Séjour on (codeLogement = codeLogement)
```

Solution : le renommage des relations.

```
select *
from   Activité as A
join   Séjour as S
on     (A.codeLogement = S.codeLogement)
```

Composition des jointures

On peut placer des expressions algébriques quelconques dans le `from`. Ici, deux jointures $(V \bowtie S) \bowtie L$

```
select nom, nomLogement
from (Voyageur as V
      join Séjour as S on V.id = S.idVoyageur) as VS
      join Logement as L on VS.codeLogement = L.code
```

Lisibilité aléatoire...À comparer avec la version déclarative.

```
select V.nom as nom, L.nom as nomLogement
from   Voyageur as V, Séjour as S, Logement as L
where  V.id = S.idVoyageur
and    S.codelogement = L.code
```

La condition de jointure

$$R \underset{R.A\ \Theta\ S.B}{\bowtie} S, \quad \Theta \in \{=, <, >, \leq, \geq\}$$

Toute comparaison d'attributs de R et de S peut apparaitre dans la condition de la jointure.

```
select *
from Logement as L
join Activité as A
on   (L.code >= A.codeLogement)
```

On la nomme parfois **Θ–jointure**, par extension de l'équi-jointure.

À retenir

L'algèbre est un langage pour créer des relations à partir d'autres relations. La traduction en SQL consiste à placer des expressions dans la clause `from`, de façon imbriquée.

- La jointure est une opération courante, essentielle et sensible (performances)
- Elle peut s'exprimer littéralement en SQL avec `join ... on ...`
- Ce n'est qu'une version alternative à l'expression équivalente de la syntaxe déclarative de SQL

Expressions algébriques (S4.3)

Algèbre relationnelle : expressions algébriques

Par composition on exprime avec l'algèbre toutes les requêtes relationnelles que l'on peut aussi exprimer – plus naturellement – avec la logique.

Dans cette section, des exemples d'expressions algébriques courantes

- Composition de sélections
- Jointures
- Différence
- Et même la quantification universelle

Composition de sélections

La conjonction de critères s'obtient en composant des sélections.

$$\sigma_{\texttt{capacité}>100}(\sigma_{\texttt{type}=\text{'Hôtel'}}(\texttt{Logement}))$$

On acceptera de l'écrire

$$\sigma_{\texttt{capacité}>100 \,\wedge\, \texttt{type}=\text{'Hôtel'}}(\texttt{Logement}))$$

L'union est l'équivalent en algèbre de la disjonction logique.

$$\sigma_{\texttt{capacité}>100}(\texttt{Logement}) \cup \sigma_{\texttt{lieu}=\text{'Corse'}}(\texttt{Logement})$$

Peut s'écrire

$$\sigma_{\texttt{capacité}>100 \,\vee\, \texttt{lieu}=\text{'Corse'}}(\texttt{Logement})$$

Composition de sélections, suite

La différence permet d'exprimer l'opérateur $\neq$.

$$\sigma_{\texttt{capacité}>100}(\texttt{Logement}) - \sigma_{\texttt{lieu}=\text{'Corse'}}(\texttt{Logement})$$

On acceptera de l'écrire

$$\sigma_{\texttt{capacité}>100 \,\wedge\, \neg\texttt{lieu}=\text{'Corse'}}(\texttt{Logement})$$

En résumé, on peut écrire la sélection en exprimant les critères avec une formule F du calcul propositionnel, $\sigma_F(R)$.

Attention à la négation : voir plus loin.

Requêtes conjonctives

Toute requête s'écrivant avec σ, π, $\times$ (et donc $\bowtie$).

Requêtes mono-table

Nom des logements en Corse

$$\pi_{\texttt{nom}}(\sigma_{\texttt{lieu='Corse'}}(\texttt{Logement}))$$

Nom et prénom des clients corses

$$\pi_{\texttt{nom,prénom}}(\sigma_{\texttt{région='Corse'}}(\texttt{Voyageur}))$$

Requêtes conjonctives, suite

Requêtes multi-tables, avec jointure.

Nom des clients qui sont allés à Tabriz

$$\pi_{\texttt{nom}}\left(\texttt{Voyageur} \underset{\texttt{id=idVoyageur}}{\bowtie} (\sigma_{\texttt{codeLogement='ta'}}(\texttt{Séjour}))\right)$$

Quels lieux a visité le client 30

$$\pi_{\texttt{lieu}}\left(\sigma_{\texttt{idVoyageur}=30}(\texttt{Séjour}) \underset{\texttt{codeLogement=code}}{\bowtie} \texttt{Logement}\right)$$

Requêtes conjonctives, suite de suite

Nom des clients qui ont eu l'occasion de faire de la voile. En deux étapes.

$$Q_1 := \texttt{Séjour} \underset{\texttt{codeLogement=codeLogement}}{\bowtie} (\sigma_{\texttt{codeActivité='Voile'}}(\texttt{Activité}))$$

puis

$$\pi_{\texttt{nom}}(\texttt{Voyageur} \underset{\texttt{id=idVoyageur}}{\bowtie} Q_1)$$

Extension de la condition de jointure

La conjonction de critères s'obtient en composant la jointure avec la sélection.

$$\sigma_{\neg \text{code}=\text{codeLogement}}(\text{Logement} \underset{\text{nom}\geq\text{codeActivité}}{\bowtie} \text{Activité}) = \text{Logement} \underset{\neg\text{code}=\text{codeLogement} \wedge \text{nom}\geq\text{codeActivité}}{\bowtie} \text{Activité}$$

```
select *
from   Logement as L
join   Activité as A
on     (not L.code = A.codeLogement and L.nom >= A.codeActivité)
```

Toute combinaison de conditions liées par des connecteurs logiques est éligible.

La jointure naturelle : un cas singulier

Soient $R(A, B, C)$ et $S(B, C, D)$; la jointure naturelle de R et S est définie ci-dessous.

$$R \bowtie S = \pi_{A,R.B,R.C,D}(\sigma_{R.B=S.B \wedge R.C=S.C}(R \times S))$$

Les deux particularités

- Les attributs communs de R et de S font partie de la condition de jointure par égalité
- Seule une occurrence de chaque paire d'attributs figure dans le schéma du résultat.

```
select       *
from         Séjour as S
natural join Logement as L   -- jointure sur S.nom = L.nom
```

La négation

S'exprime en algèbre avec la différence.

Codes des logements qui ne proposent pas de voile

$$\pi_{\texttt{code}}(\texttt{Logement}) - \rho_{\texttt{codeLogement} \to \texttt{code}}\left(\pi_{\texttt{codeLogement}}(\sigma_{\texttt{codeActivité}=\text{'Voile'}}(\texttt{Activité}))\right)$$

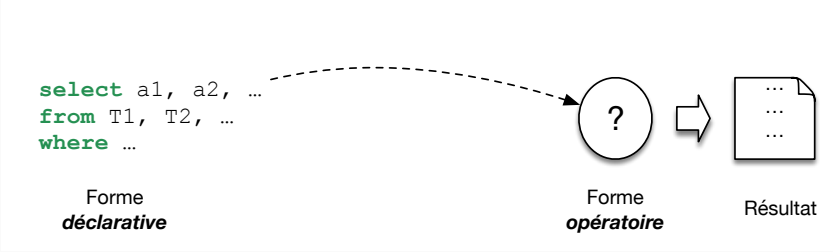
Raisonnement :

Je prends tous les logements moins ceux qui proposent de la voile.

Peu pratique si on compare au `not exists` de SQL : la différence ne s'applique que sur des relations ayant le même schéma.

Logique ou fonctionnel, quel langage ?

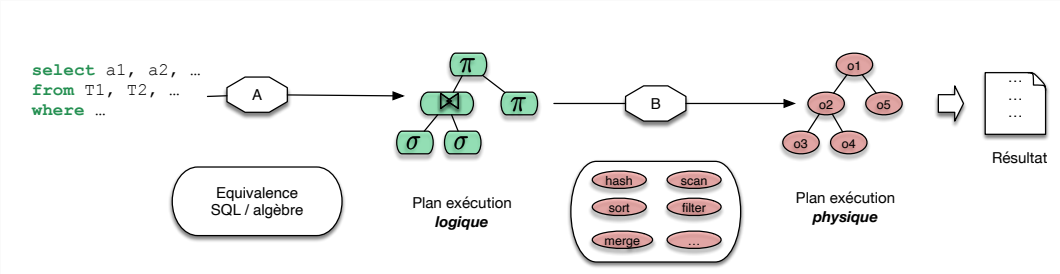
On soumet une requête SQL, et après ?



On exprime la requête de manière déclarative, le système doit trouver une méthode d'évaluation.

Expression logique, évaluation fonctionnelle

Le système peut produire une expression algébrique pour toute requête SQL.



Il reste à trouver les bons algorithmes, à utiliser correctement les ressources de calcul : un défi pour le système, pour l'utilisateur avisé et pour l'administrateur.

À retenir

L'algèbre est un langage relationnel équivalent à la logique formelle.

- On peut exprimer toutes les requêtes en algèbre, par composition d'opérateurs
- Il existe une syntaxe SQL pour toutes les requêtes algébriques
- Le style est celui de la programmation (fonctionnelle) ; moins clair, moins accessible à un non-programmeur, moins proche de l'expression « naturelle » de la requête
- Peut s'effectuer en plusieurs étapes pour une meilleure clarté.

L'algèbre était conçue à l'origine pour définir l'exécution des requêtes, et non leur formulation.