SQL et les bases de données relationnelles

SQL déclaratif : le calcul relationnel

Guillaume Raschia — Nantes Université

originaux de Philippe Rigaux, CNAM

Dernière mise-à-jour : 17 novembre 2025

Plan de la session

SQL conjonctif (S3.2)

Quantification et négation (S3.3)

Conception de requêtes (S3.4)

SQL conjonctif (S3.2)

SQL, première partie

Cette section présente les requêtes SQL conjonctives qui sont celles qui s'expriment sans négation (¬) ni disjonction (∨).

- Forme d'une requête SQL : variables-nuplets, conditions, construction du nuplet-résultat
- Requêtes mono-variables
- Requêtes multi-variables

Variable-nuplet

SQL manipule des nuplets ouverts de la forme $t = (a_1, a_2, \cdots, a_n)$.

Nous les appellerons des variables-nuplets.

La quantification porte sur la variable-nuplet t : $\exists t$ et $\forall t$.

On désigne les attributs en les rattachant à t : $t.a_1$, $t.a_2$, etc.

On peut exprimer des comparaisons : $t.a_i = \text{a}$ ou $t.a_i = t.a_j$

Requête mono-variable

Les requêtes les plus simples utilisent une seule variable-nuplet. Leur forme logique du calcul relationnel est :

$$\{t.a_1, t.a_2, \cdots, t.a_n \mid T(t) \wedge F_{\text{cond}}(t)\}$$

ou de manière équivalente, avec des variables-domaines :

$\{x_1, x_2, \cdots, x_n \mid \exists \vec{y},\ T(\vec{x}, \vec{y}) \wedge F_{\text{cond}}(\vec{x}, \vec{y})\}$ ou $Q(\vec{x}) = \exists \vec{y},\ T(\vec{x}, \vec{y}) \wedge F_{\text{cond}}(\vec{x}, \vec{y})$

Requête mono-variable : traduction en SQL

Expression du calcul relationnel :

$$\{t.a_1, t.a_2, \cdots, t.a_n \mid T(t) \wedge F_{\text{cond}}(t)\}$$

La forme SQL

```
select [distinct] t.a1, t.a2, ..., t.an
from   T as t
where  <condition>
```

C'est un « bloc » avec trois clauses :

- le `from` définit la variable libre et sa portée
- le `where` définit la condition sur la variable libre
- le `select` (avec `distinct` optionnel) construit le nuplet-résultat

Parlons du `distinct`

Une relation n'a pas de doublon. Or certaines requêtes peuvent en produire :

```
select l.type from Logement as l
```

type
Auberge
Hôtel
Gîte
Hôtel

Le `distinct` garantit que les doublons sont éliminés.

```
select distinct l.type from Logement as l
```

Certaines requêtes ne peuvent pas produire de doublon ! À approfondir.

Premier exemple

Code, nom et type des logements en Corse.

```
select t.code, t.nom, t.type
from   Logement as t
where  t.lieu = 'Corse'
```

Correspond à la formule

$$\{t.\texttt{code}, t.\texttt{nom}, t.\texttt{type} \mid \texttt{Logement}(t) \wedge t.\texttt{lieu} = \text{'Corse'}\}$$

Forme simplifiée :

```
select code, nom, type
from   Logement
where  lieu = 'Corse'
```


Interprétation

La variable est affectée à chaque nuplet de la table définie par la portée.

On garde toutes les affectations qui satisfont la condition F_{cond}.

La seule affectation correcte est surlignée ci-dessous.

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

Trivial ? Oui,
et tant mieux, car cette interprétation fonctionne pour toutes les requêtes.

Requête multi-variables

Regardons pour deux variables : la généralisation est facile.

Forme de la requête :

```
select [distinct] t1.a1, ..., t1.an, t2.a1, ..., t2.am
from T1 as t1, T2 as t2
where <condition>
```

Interprétation

Parmi toutes les affectations possibles des variables, on ne conserve que celles qui satisfont la condition exprimée par F_{cond}.

Un exemple détaillé : logements où on peut pratiquer la plongée

$$\{\ell.\texttt{code}, \ell.\texttt{nom}, a.\texttt{codeActivité} \mid \texttt{Logement}(\ell) \wedge \texttt{Activité}(a) \wedge a.\texttt{codeActivité} = \text{'Plongée'} \wedge \ell.\texttt{code} = a.\texttt{codeLogement}\}$$

Nous avons besoin de deux variables a et ℓ :

- la première (a) parcourt les nuplets de `Activité` ;
- la seconde (ℓ) parcourt les nuplets de `Logement` ;
- l'attribut `codeActivité` de a vaut « Plongée » ;
- les deux variables partagent le même code de logement.

```
select distinct l.code, l.nom, a.codeActivité
from   Logement as l, Activité as a
where  l.code = a.codeLogement
and    (a.codeActivité = 'Plongée' or a.codeActivité = 'Voile')
```

Interprétation : affectation des deux variables

Logement (variable ℓ)

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

Activité (variable a)

codeLogement	codeActivité
ca	Randonnée
ta	Plongée
ge	Ski
pi	Plongée
pi	Voile

Deuxième exemple : les paires de logements qui sont du même type

Nous avons besoin de deux variables,

- chacune ayant pour portée la table `Logement`
- les deux variables partagent le même attribut `type`

```
select distinct l1.nom as nom1, l2.nom as nom2
from Logement as l1, Logement as l2
where l1.type = l2.type
```

Soit la formule

$$\{\ell_1.\texttt{nom}, \ell_2.\texttt{nom} \mid \texttt{Logement}(\ell_1) \wedge \texttt{Logement}(\ell_2) \wedge \ell_1.\texttt{type} = \ell_2.\texttt{type}\}$$

ou de manière équivalente :

$$\{x, y \mid \exists z, \texttt{Logement}(\_, x, \_, z, \_) \wedge \texttt{Logement}(\_, y, \_, z, \_)\}$$

Interprétation : affectation des deux variables

Logement (variable ℓ_1)

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

Logement (variable ℓ_2)

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

Autre affectation possible

Logement (variable ℓ_1)

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

Logement (variable ℓ_2)

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

Encore une autre (et trois autres encore possibles)

Logement (variable ℓ_1)

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

Logement (variable ℓ_2)

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

À retenir

Quelle que soit sa complexité, l'interprétation d'une requête SQL peut toujours se faire de la manière suivante.

- Chaque variable du `from` peut être affectée à tous les nuplets de sa portée.
- Le `where` définit une condition sur ces variables : seules les affectations satisfaisant cette condition sont conservées.
- Le nuplet résultat est construit à partir de ces affectations.

Quantification et négation (S3.3)

SQL, quantificateurs et négation

Cette section présente les quantifications existentielle et universelle.

Les quantificateurs permettent l'expression de la négation : « je veux tous ces nuplets sauf ceux-là ».

Dans cette section :

- Le quantificateur `exists` et l'expression du « pour tout »
- La négation
- Equivalence des requêtes : plusieurs syntaxes, une seule interprétation.

Le quantificateur “exists“

Considérons la requête « les logements où l'on peut faire du ski ».

$$\{\ell.\texttt{nom} \mid \texttt{Logement}(\ell) \wedge \exists a, \texttt{Activité}(a) \wedge a.\texttt{codeActivité} = \text{'Ski'} \wedge \ell.\texttt{code} = a.\texttt{codeLogement}\}$$

En SQL :

```
select distinct l.nom
from   Logement as l, Activité as a
where  l.code = a.codeLogement
and    a.codeActivité = 'Ski'
```

La variable a n'intervient pas dans le nuplet-résultat. Il s'agit d'une variable liée.

Suite de l'exemple

Légère reformulation : on cherche les logements où il existe une activité “Ski”.

requête SQL équivalente

```
select distinct l.nom
from   Logement as l
where  exists (select 42
               from   Activité as a
               where  l.code = a.codeLogement
               and    a.codeActivité = 'Ski')
```

Bien que littérale, l'imbrication n'est pas recommandée là où il existe une version « à plat » de la requête.

Construction de formules complexes

On peut construire des formules imbriquées (sous-requêtes SQL) sans limitation de profondeur.

Qui est allé dans les Alpes ?

```
select distinct v.prénom, v.nom
from   Voyageur as v, Séjour as s, Logement as l
where  v.id = s.idVoyageur
and    s.codeLogement = l.code
and    l.lieu = 'Alpes'
```

Ni s ni ℓ ne sont utilisées dans la construction du nuplet-résultat.

Qui est allé dans les Alpes ?

Avec un quantificateur existentiel : « Les voyageurs pour lesquels il existe un séjour dans les Alpes ».

```
select distinct v.prénom, v.nom
from   Voyageur as v
where  exists (select 42
               from   Séjour as s, Logement as l
               where  v.id = s.idVoyageur
               and    s.codeLogement = l.code
               and    l.lieu = 'Alpes')
```

Plus clair ? Moins clair ?

Imbrication d'imbrication

Avec deux quantificateurs existentiels : « Les voyageurs pour lesquels il existe un séjour dont le logement existe dans les Alpes ».

```
select distinct v.prénom, v.nom
from   Voyageur as v
where  exists (select 42
               from   Séjour as s
               where  v.id = s.idVoyageur
               and    exists (select 42
                              from   Logement as l
                              where  s.codeLogement = l.code
                              and    l.lieu = 'Alpes')
              )
```

Pas très naturel.

Négation

Les logements qui ne proposent pas de Ski.

$$\{\ell.\texttt{nom} \mid \texttt{Logement}(\ell) \wedge \neg\exists a, \texttt{Activité}(a) \wedge a.\texttt{codeActivité} = \text{'Ski'} \wedge \ell.\texttt{code} = a.\texttt{codeLogement}\}$$

Correspond à la formulation : Les logements où il n'existe pas d'activité Ski.

```
select distinct l.nom
from   Logement as l
where  not exists (select 42
                   from   Activité as a
                   where  l.code = a.codeLogement
                   and    a.codeActivité = 'Ski')
```


Quantificateur universel

Les voyageurs qui ont séjournés dans tous les logements.

$$\{v.\texttt{prénom}, v.\texttt{nom} \mid \texttt{Voyageur}(v) \wedge \forall \ell,\ \texttt{Logement}(\ell) \rightarrow (\exists s,\ \texttt{Séjour}(s) \wedge s.\texttt{codeLogement} = \ell.\texttt{code} \wedge s.\texttt{idVoyageur} = v.\texttt{id})\}$$

La formule implicative est « intuitive ».

Ni l'implication, ni le quantificateur universel ne sont traduisibles en SQL.

- L'implication : $p \rightarrow q \ \Leftrightarrow\ \neg p \vee q$
- Le quantificateur universel : $\forall x, P(x) \ \Leftrightarrow\ \neg\exists x, \neg P(x)$

Quantificateur universel par la double négation

Reformulation avec double négation : on cherche les voyageurs pour lesquels il n'existe pas de logement dans lequel ils n'ont pas séjourné.

```
select distinct v.prénom, v.nom
from   Voyageur as v
where  not exists (select 42
                   from   Logement as l
                   where  not exists (select 42
                                      from   Séjour as s
                                      where  l.code = s.codeLogement
                                      and    v.id = s.idVoyageur)
```

À retenir

SQL = un langage normalisé à la définition très précise.

À propos de la construction `SELECT-FROM-WHERE`

- Tout ce qui peut s'exprimer par une formule logique est exprimable en SQL. Ni plus, ni moins.
- Inversement, tout ce qui ne s'exprime pas par une formule (boucles, incrémentations, etc.) ne s'exprime pas en SQL.

Maîtriser SQL = savoir formuler sa requête de manière rigoureuse.

Conception de requêtes (S3.4)

Construction d'une requête SQL

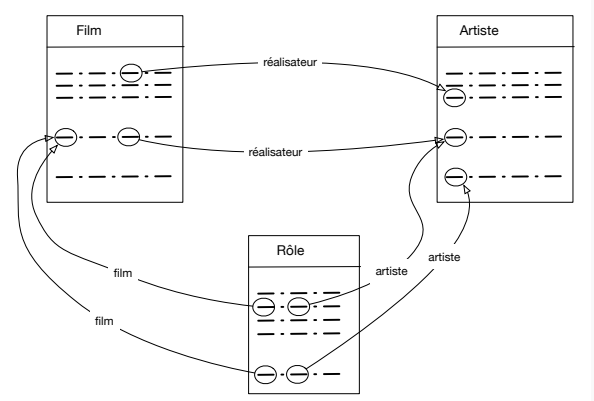
- Le résultat d'une requête est une relation constituée de nuplets.
- Chaque nuplet du résultat est construit à partir d'un ensemble de n nuplets $t_1, t_2, \cdots, t_n$ provenant de la base de données.
- Ces n nuplets doivent satisfaire un ensemble de conditions (exprimé par une formule).

La clause `from` sert à définir les $t_1, t_2, \cdots, t_n$, la clause `where` à définir les conditions, la clause `select` à construire un nuplet-résultat à partir des $t_1, t_2, \cdots, t_n$.

Dans cette section nous étudions le processus (mental) de conception d'une requête.

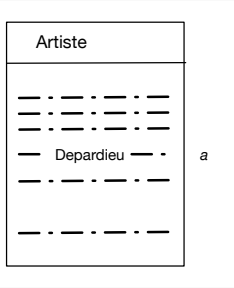
Étape préalable : comprendre le schéma

Il faut savoir visualiser les tables, et les liens entre ces tables.



Imaginer une requête = visualiser les nuplets

Exemple trivial :
“Quel âge a Gérard Depardieu ?”



On identifie un nuplet, a, qui suffit.

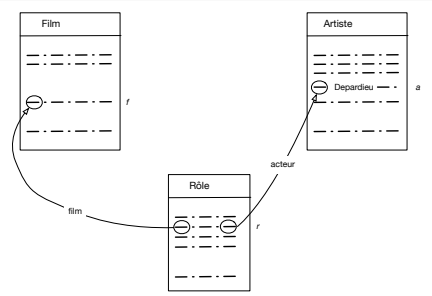
La requête est une transcription directe de la visualisation.

```
select annéeNaissance
from   Artiste as a
where  a.nom='Depardieu'
```

Pour l’instant tout est simple.

Souvent il faut plusieurs nuplets

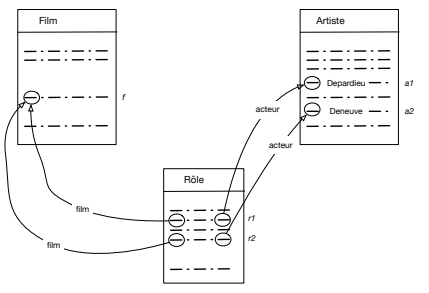
Exemple : les films avec Gérard Depardieu ?




```
select f.titre
from   Artiste as a,
       Rôle as r,
       Film as f
where  a.nom = 'Depardieu'
and    a.idArtiste = r.idActeur
and    r.idFilm = f.idFilm
```

La requête est encore une transcription directe de la visualisation.

Les films avec C. Deneuve et G. Depardieu




```
select *
from   Artiste as a1,
       Artiste as a2,
       Rôle as r1,
       Rôle as r2,
       Film as f
where  a1.nom = 'Depardieu'
and    a2.nom = 'Deneuve'
and    a1.idArtiste = r1.idActeur
and    a2.idArtiste = r2.idActeur
and    r1.idFilm = f.idFilm
and    r2.idFilm = f.idFilm
```


À retenir

La démarche mentale pour construire une requête SQL est (toujours) la suivante.

- On détermine les nuplets (et surtout leurs tables) nécessaires pour construire un nuplet du résultat.
  ⇒ ça définit le `from`
- On détermine les conditions que doivent satisfaire ces nuplets.
  ⇒ ça définit le `where`

  Important : une condition peut être définie par une sous-requête (résultat vide ou non)
- Il ne reste plus qu’à « piocher » dans les nuplets pour constituer le résultat
  ⇒ ça définit le `select` et complète la requête.