Bases de données

Quelques rappels de SQL

stephane@gonnord.org - http://blog.psi945.fr

Lundi 29 février 2016

Plan

Communes, régions et départements

commune			
id	dep	nom	pop
...	...	...	...
69023	69	Lyon	484344
...	...	...	...
2BO50	2B	Calvi	5394
...	...	...	...

departement		
id	reg	nom
...	...	...
69	82	Rhône
...	...	...
2B	94	Haute-Corse
...	...	...

region	
id	nom
...	...
82	Rhône-Alpes
...	...
94	Corse
...	...

Élèves, colleurs et colles

eleves		
id	nom	prenom
0	Lyons	Jacques-Louis
1	Laurent	Jean
...	...	...

profs		
id	nom	prenom
0	Théron	Pierre
1	Brun	Jules
...	...	...

colles			
prof	eleve	semaine	note
2	8	1	16
1	0	6	19
...	...	...	...

Format général

```
SELECT <expressions>
FROM <tables>
WHERE <conditions>
GROUP BY <attributs>
HAVING <conditions>
ORDER BY <attributs>
```

Dans telles tables, tu prends les lignes vérifiant telles conditions ; tu les groupes selon tels critères. Dans les groupes, tu va juste prendre ceux dont telle moyenne (par exemple) sur tel attribut vérifie telle condition. Ah et puis tu vas me donner le résultat sous forme triée selon tels attributs !

Un exemple

Wow !

```
SELECT eleve, eleves.nom, COUNT(*) AS plantages
FROM eleves JOIN colles
ON eleves.id = colles.eleve
WHERE note<8
GROUP BY eleve
HAVING plantages>=5
ORDER BY plantages
```

Alors, que vient-on de demander ?

Don’t panic... on va commencer par des choses plus simples !

Sélection, projection

- Requête de base :

```
SELECT <tels attributs>
FROM <telle table>
WHERE <telle(s) condition(s)>
```

- Exemples :
  - ```
    SELECT *
    FROM communes
    ```
  - ```
    SELECT nom,pop
    FROM communes
    ```
  - ```
    SELECT nom,pop
    FROM communes
    WHERE pop > 100000
    ```
  - ```
    SELECT *
    FROM triangles
    WHERE ab = bc AND bc = ac
    ```

Opérations ensemblistes

- Union, intersection, différence : Bof...
  - Rarement utile (sauf dans les exos dédiés...)
  - Souvent déconnant (torchons, serviettes...)
- SQL : syntaxe moisie...
  - `SELECT ... FROM ... UNION SELECT ... FROM ...`
  - `SELECT ... FROM ... INTERSECT SELECT ... FROM ...`
  - `SELECT ... FROM ... EXCEPT SELECT ... FROM ...`
- Et si les attributs sont différents ? *Je ne veux même pas savoir ce qui se passe !*
- Produit cartésien :
  - Définition : comme en maths :

$$A \times B = \{(a, b) \mid a \in A \text{ et } b \in B\}$$

  - SQL : pas comme en maths !

```
SELECT * from table1 , table2
```

De l'inutilité du produit cartésien

- Deux tables :

commune	
nom	dep
Lyon	69
Calvi	2B
Corte	2B

departement	
id	nom
69	Rhône
2B	Haute-Corse

- Et leur produit :

commune × departement			
nom	dep	**id**	nom
Lyon	69	69	Rhône
Lyon	69	2B	Haute-Corse
Calve	2B	69	Rhône
Calvi	2B	2B	Haute-Corse
Corte	2B	69	Rhône
Corte	2B	2B	Haute-Corse

Jointure

- Deux tables :

commune	
nom	dep
Lyon	69
Calvi	2B
Corte	2B

departement	
id	nom
69	Rhône
2B	Haute-Corse

- Et leur *jointure* naturelle :

commune ⋈ departement			
nom	dep	**id**	nom
Lyon	69	69	Rhône
Calvi	2B	2B	Haute-Corse
Corte	2B	2B	Haute-Corse

- Une base de données est *pensée* dès le départ autour des jointures de tables, et les clés le permettant.

Jointure

- Formellement : $R_1 \underset{a=b}{\bowtie} R_2 = \{(x, y) \in R_1 \times R_2;\ x.a = y.b\}$...
- Et si R_1 et R_2 ont deux attributs de même nom ?
- En SQL : deux syntaxes équivalentes :
  - ```
    SELECT ...
    FROM table1 JOIN table2
    ON condition
    WHERE ...
    ```
  - ```
    SELECT ...
    FROM table1 , table2
    WHERE condition AND ...
    ```
- SQL, encore :
  - ```
    WHERE table1.foo = table2.bar
    ```
  - ```
    SELECT ...
    FROM table1 JOIN table2 JOIN table3
    ON condition1 AND condition2
    WHERE ...
    ```

Clés primaires

- *Clé primaire* : un (ensemble d')attribut(s) caractérisant les éléments d'une table (injectivité de $x \mapsto x.c$).
- Bien avoir le schéma relationnel devant les yeux.
- Exemples :
  - ```
    FROM communes JOIN departements
    ON communes.dep = departements.id
    ```
  - ```
    FROM eleves JOIN colles
    ON ide = eleve
    ```
  - ```
    FROM eleves JOIN colles JOIN profs
    WHERE ide = eleve AND prof = idp
    ```
  - ```
    FROM clubs c1 JOIN matchs JOIN clubs c2
    ON c1.idc = eq1 AND c2.idc=eq2
    ```

Agrégats

- Principe :
  - On regroupe (en général) les lignes suivant des attributs ;
  - on applique une *fonction d'agrégation* à chacun de ces groupes.
- Fonctions : `MIN, MAX, COUNT(...), AVG, SUM(...)`
- SQL : un exemple interdit/scandaleux/absurde.

```
SELECT reg,departements.nom,count(*)
FROM departements JOIN regions
ON departements.reg = regions.id
GROUP BY reg
```

reg	nom	count(*)
1	Guadeloupe	1
11	Seine-et-Marne	8
...	...	...

Agrégats

- Condition en amont (`WHERE`) et/ou aval (`HAVING`)

```
SELECT reg, regions.nom, count(*)
FROM departements JOIN regions
ON departements.reg = regions.id
WHERE departements.id <= 34
GROUP BY reg
HAVING count(*) >= 3
```

reg	nom	count(*)
73	Midi-Pyrénées	3
91	Languedoc-Roussillon	3

- À retenir :

On projette les attributs selon lesquels on a groupé les lignes...
Et a priori aucun autre

Un algorithme de conception de requêtes

Plusieurs passages sont possibles...

1. `SELECT` : quels attributs (et/ou agrégats) nous intéressent ?
2. `FROM` : issus de quelles tables ?
3. `JOIN ... ON` : si on joint n tables, il y a *a priori* $n-1$ conditions de jointures.
4. `WHERE` : quelles conditions/restrictions en amont ?
5. `GROUP BY` : comment veut-on regrouper les tuples ? *Penser à ajouter les attributs au* `SELECT` *initial...*
6. `HAVING` : restrictions en aval, portant sur les agrégats.
7. Sous-requêtes éventuelles, paramétrées ou non.

C’est fini

Merci de votre attention