SQL vs Algèbre Relationnelle

PCSI - Lycée du Parc

-

Vocabulaire

SQL	Algèbre relationnelle
table	relation
colonne	attribut
ligne	enregistrement / tuple

SELECT colonne1, colonne2 FROM table projection :

$$\pi_{colonne1,colonne2}(table)$$

... WHERE condition1 AND condition2 selection, restriction :

$$\sigma_{condition1 \wedge condition2}(R)$$

SELECT col1, col2, f(col3), g(col4) GROUP BY col1, col2

agrégation : $_{col1,col2}\gamma_{f(col3),g(col4)}(R)$

ORDER BY col ASC, ORDER BY col DESC pas en A.R.

table1 JOIN table2 ON table1.a = table2.b jointure :

$$table1 \underset{a=b}{\bowtie} table2$$

Fonctions (avec agrégation) : MIN(...), MAX(...), SUM(...), AVG(...), COUNT(...), COUNT (DISTINCT ...)

On peut aussi faire des calculs : additions, multiplications...

Conditions utilisables dans WHERE

- WHERE attr = “Bonjour” (test d’églité)
- WHERE attr LIKE “Bon%ur” (“Bonjour”, “Bonheur”, . . .)
- WHERE attr = attr2
- WHERE attr IN colonne

```
SELECT colles.note
FROM colles
WHERE colles.ide = (SELECT id FROM eleves
                    WHERE nom="Morgan" AND prenom="Dexter")

SELECT eleves.prenom
FROM eleves
WHERE eleves.id IN (SELECT ide
                    FROM colles
                    GROUP BY ide
                    HAVING AVG(note) >= 10)
```

Schéma relationnel

Clé primaire / Clé étrangère

Exemple 1 :

- eleves (<u>ide</u>, nom)
- colles (eleve, prof, note)

Exemple 2 :

- villes (dep, nom_ville, pop)

(ou faire un vrai schéma !)

Requêtes sans agrégation

$$\pi_{attributs}(\sigma_{conditions}(matable))$$

```
SELECT attributs
FROM matable
WHERE conditions
```

$$\sigma_{conditions}(matable)$$

```
SELECT *
FROM matable
WHERE conditions
```

Avec agrégation

$$\pi_{\ldots}(\sigma_{postconditions}({}_{attributs}\gamma_{calculs}(\sigma_{precondtions}(R))))$$

```
SELECT dep, SUM(pop) AS s
FROM villes
WHERE pop > 10000
GROUP BY dep
HAVING s < 100000
```

Pourquoi ça n'a pas de sens de garder les autres colonnes ?

Construction plus compliquée

Inner JOIN

```
SELECT a.nom, b.nom
FROM matchs
     JOIN clubs AS a ON matchs.equipe1 = a.id
     JOIN clubs AS b ON matchs.equipe2 = b.id
```

Requêtes imbriquées

- WHERE ... = (SELECT ...)
- WHERE ... IN (SELECT ...)
- FROM (SELECT ...)
- FROM table1 JOIN (SELECT ...) ON ... = ...

Exemples : MAX

$$\gamma_{MAX(pop)}(villes)$$

```
SELECT MAX(pop)
FROM villes
```

Problème A : Calculer la taille de la plus grosse ville dans chaque département : facile

Solution A :

```
SELECT dep, MAX(pop)
FROM villes
GROUP BY dep
```

Exemples : MAX

Problème B : Calculer la taille de la plus grosse ville (dans toute la France) et le nom de la/les villes associées

Solution B0 : (attention, ne fonctionne pas ! pourquoi ?)

```
SELECT nom_ville, MAX(pop)
FROM villes
```

Solution B1 : (plusieurs résultats possibles)

```
SELECT nom_ville, pop
FROM villes
WHERE pop = (SELECT MAX(pop) AS max_pop FROM villes)
```

Solution B2 : (un seul résultat)

```
SELECT nom_ville, pop
FROM villes
ORDER BY pop DESC LIMIT 1
```

Exemples : MAX

Problème C : Calculer la taille de la plus grosse ville de chaque département et le nom de la/les villes associées

```
SELECT *
FROM villes JOIN (SELECT dep AS dep2 , MAX(pop) AS max_pop
                  FROM villes
                  GROUP BY dep2)
     ON dep = dep2
WHERE pop = max_pop
```

Exemples : MAX

Problème D : Le nom des villes qui ont la deuxième plus grande taille...

```
SELECT nom
FROM villes
WHERE pop = (SELECT MAX(pop) AS deuxieme_plus_grand
             FROM villes
             WHERE pop <> (SELECT MAX(pop) AS plus_grand
                           FROM villes))
```

S'il n'y a pas deux villes avec la plus grande taille

```
SELECT nom
FROM villes
ORDER BY pop DESC
LIMIT 1 OFFSET 1 -- On oublie la première ligne
```

Dans Python

```
import sqlite3
conn = sqlite3.connect('example.db')
c = conn.cursor()

for row in c.execute('SELECT * FROM stocks ORDER BY price'):
        print row

(u'2006-01-05', u'BUY', u'RHAT', 100, 35.14)
(u'2006-03-28', u'BUY', u'IBM', 1000, 45.0)
(u'2006-04-06', u'SELL', u'IBM', 500, 53.0)
(u'2006-04-05', u'BUY', u'MSFT', 1000, 72.0)
```