Bases de données - Langage SQL

Le but est de stocker des données de sorte à pouvoir effectuer facilement et rapidement des recherches et à minimiser la mémoire utilisée : la présence de plusieurs tables (ou tableaux) de données au lieu d'une seule permet d'éviter des redondances et donc de gagner en mémoire. Certaines colonnes d'une table renvoient à des colonnes d'autres tables : ce sont ces références qui permettront ensuite de faire des recherches croisées sur les différentes tables simultanément.
Terminologie :
Les attributs sont les caractéristiques des données à saisir. Ce sont les titres des colonnes des tables. Un identifiant (ou clé primaire) est un attribut ou un ensemble d'attributs dont la valeur détermine les autres attributs de la table. Un enregistrement (ou valeur) d'un attribut correspond aux éléments placés dans la colonne associée.

1) **Projections :** `SELECT [DISTINCT] ... FROM ... [ORDER BY ...] [AS ...]`

```
SELECT attribut1, attribut2, ... , [nouvel_attribut AS ...]
     [liste et/ou fonctions d'attributs, par exemple une somme de 2 attributs]
FROM nom_table
WHERE conditions #
```
(définies avec `OR`, `AND`, `NOT`, `IN`, ...)

Exemple (utilise les tables définies au paragraphe 6)) :

```
SELECT nom, age
FROM zoo
WHERE espece = 'girafe' AND age > 20
```

Remarque :

La sélection `SELECT` est à voir comme une projection sur un sous-ensemble des colonnes (on ne conserve que certaines colonnes), alors que la condition `WHERE` est une sélection d'un sous-ensemble de lignes (celles qui vérifient les conditions mentionnées).

Remarque : `SELECT` ne supprime pas les redondances dans les réponses.
On peut le forcer à le faire en ajoutant l'option `DISTINCT`.

On peut aussi classer les éléments selon un ordre défini par l'un des attributs :
`SELECT DISTINCT nom, age FROM table ORDER BY age`

Remarque : Dans le cas de chaînes de caractères, c'est l'ordre alphabétique qui est pris en compte.

On peut aussi (re)nommer les attributs : `SELECT nom AS surnom , age FROM zoo`

2) **Agrégations :** [`COUNT(), AVG(), MIN(), MAX()`] `[AS ...] ... GROUP BY... [HAVING ...]`

Liste des fonctions agrégatives classiques :

`COUNT(*)` : nombre de lignes de la table , `COUNT(attribut)` : nombre de lignes remplies pour l'attribut

`AVG(attribut)` : moyenne ; `SUM(attribut)` : somme ;
`MIN(attribut)` : minimum ; `MAX(attribut)` : maximum (sélectionne *la* valeur maximale)

Les fonctions agrégatives servent à définir de nouvelles colonnes, construites à partir des valeurs des colonnes existantes. *Attention* : Les fonctions agrégatives ne peuvent pas être utilisées avec une condition.

Les fonctions agrégatives sont définies par l'instruction : `GROUP BY attribut [HAVING condition]`

qui permet de calculer les valeurs agrégatives sur les paquets de données groupés selon la valeur de l'attribut donné par `GROUP BY` en ne sélectionnant que les lignes vérifiant la condition définie par `HAVING`.

Remarque : Si on veut donner un nom à l'attribut ainsi obtenu, on utilise `AS` :

```
SELECT batiment, COUNT(nom) AS nombre
FROM bati
GROUP BY batiment
ORDER BY nombre
```

renvoie

batiment	nombre
1	3
2	4

: le numéro du batîment et le nombre de pensionnaires

Exemple :

```
SELECT nom, MAX(age)
FROM zoo
WHERE espece IN (SELECT espece FROM protect)
```

renvoie un doyen des espèces protégées.

Pour obtenir une liste d'un doyen de chaque espèce :

```
SELECT nom
FROM ( SELECT nom, espece, MAX(age)
       FROM zoo
       GROUP BY espece )
```

3) **Jointures :** `SELECT ... FROM ... JOIN ... ON ...`

Une jointure revient (d'un point de vue théorique) à une instruction de sélection sur le produit cartésien.

```
SELECT attributs
FROM table1 JOIN table2 ON table1.attribut1 = table2.attribut2
```

Exemple :

table1 :

personnages	*jouets*	*nombre*
Idefix	os	1
Idefix	balle	1
Medor	os	3
Balthazar	carotte	2
JollyJumper	carotte	3
Duchesse	souris	1

; table2 :

article	*poids*	*couleur*	*annee*
balle	200	rouge	2000
carotte	150	rouge	2020
os	100	jaune	2020
souris	200	grise	1990

```
SELECT DISTINCT table1.personnages, table1.jouets, table1.nombre, table2.annee
FROM table1 JOIN table2 ON table1.jouets = table2.article
WHERE table2.annee >= 2015 AND table2.couleur = jaune
ORDER BY table2.annee
```

donne :

personnages	*jouets*	*nombre*	*annee*
Idefix	os	1	2020
Medor	os	3	2020

Remarque : Lorsqu'il n'y a pas d'ambiguité sur les noms des attributs, il n'est pas nécessaire de mentionner la table concernée. On peut donc dans l'exemple ci-dessus remplacer `table1.jouets` par `jouets`, etc ...

Exemple : Pour déterminer la liste des personnages classés selon la somme des poids de leurs jouets :

```
SELECT personnages
FROM (SELECT personnages, SUM(nombre*poids) AS maxi
     FROM table1 JOIN table2 ON jouets = article
```

```
    GROUP BY personnages)
ORDER BY maxi
```

Remarque : On peut effectuer la jointure sur plusieurs tables simultanément. *Exemple* :

```
SELECT attributs
FROM table1
JOIN table2 ON table1.a1 = table2.a2
JOIN table3 ON table1.a3 = table3.a4
```

Important : Il est possible de faire une jointure d'une table (contenant ici des couples (x, y) représentant par exemple les arêtes d'un graphe) sur elle-même (en utilisant des alias). Pour déterminer les couples (x, z) pour lesquels qu'il existe y tels que (x, y) et (y, z) appartiennent à `table` :

```
SELECT t1.x , t2.y
FROM table AS t1 JOIN table AS t2
ON t1.y = t2.x
```

4) Fonctions ensemblistes

L'union et l'intersection de deux tables sont possibles lorsque les attributs sont en même nombre et de même type. Dans le cas de l'union, on obtient des enregistrements présents dans l'une ou l'autre de ces tables. L'union ne conserve que les attributs distincts.

```
SELECT * FROM table1 UNION SELECT * FROM table2
SELECT * FROM table1 INTERSECT SELECT * FROM table2
SELECT * FROM table1 EXCEPT SELECT * FROM table2
```

5) Utilisation de `DISTINCT, ORDER BY (DESC) et LIMIT ... OFFSET ...`

```
SELECT * FROM table ORDER BY attribut DESC
```

Pour limiter laffochage de la liste des enregistrements ordonnés, on utilise

```
LIMIT 3 OFFSET 5
```

Cette instruction permet de récupérer les trois premières lignes de la table en éliminant les cinq premières. Autrement dit, on obtient les lignes d'indices 6, 7 et 8.

6) Exemples

Considérons trois tables :

- la table `zoo` donne la liste des pensionnaires d'un zoo : leur nom, leur espèce, leur age, leur poids.
- la table `protect` qui indiquent la liste des espèces protégées et pour chacune le niveau de protection, les espèces non protégées ne figurant pas dans la liste).
- la table `bati` des pensionnaires et des bâtiments du zoo où ils se trouvent : une espèce peut être représentée dans plusieurs bâtiments

zoo :

nom	*espece*	*age*	*poids*
Gigi	girafe	10	1000
Baloo	chimpanzé	20	90
Jumbo	elephant	70	5000
Rose	panthere	12	35
...	...		

; protect :

espece	*niveau*
girafe	1
elephant	2
...	...

; bati :

nom	*batiment*	*clé*
Gigi	1	12
Baloo	2	15
Jumbo	1	07
...	...	

Pour obtenir la liste des espèces des animaux non protégés, classés par ordre alphabétique :

```
SELECT DISTINCT espece
FROM zoo
WHERE espece NOT IN (SELECT espece FROM protect)
ORDER BY espece
AS liste_np
```

Remarque : On utilise l'option `DISTINCT` afin que chaque espèce concernée apparaisse une seule fois.
On utilise `ORDER BY` pour obtenir la liste des espèces classées par ordre alphabétique

Pour créer une table donnant par bâtiment le nombre d'espèces protégées,
en ne gardant que les bâtiments contenant au moins deux espèces protégées :

```
SELECT batiment, COUNT (DISTINCT espece) AS nombre
FROM (SELECT batiment, espece
     FROM zoo
     JOIN bati
     ON zoo.nom = bati.nom )
GROUP BY batiment
WHERE espece IN (SELECT espece FROM protect)
HAVING nombre >= 2
```

Remarque : `HAVING` sert à sélectionner des groupes : les conditions portent sur les propriétés attribuées au groupe (définies en pratique par une ou plusieurs fonctions agrégatives). Lorsque les conditions portent sur chaque élément du groupe, il faut les spécifier par `WHERE`.

Pour obtenir UN bâtiment dont la somme des âges des pensionnaires d'espèces protégées est maximale :

```
SELECT batiment, MAX(somme)
FROM (SELECT batiment, SUM(age) AS somme
     FROM zoo
     JOIN bati
     ON zoo.nom = bati.nom
     GROUP BY batiment
     WHERE espece IN (SELECT espece FROM protect)
```

Pour obtenir LES bâtiments dont la somme des âges des pensionnaires d'espèces protégées est maximale :

```
SELECT batiment
FROM (SELECT batiment, SUM(age) AS somme
     FROM zoo
     JOIN bati
     ON zoo.nom = bati.nom
     GROUP BY batiment
     WHERE espece IN (SELECT espece FROM protect)
     AS nouvelle_table)
WHERE somme = (SELECT MAX(somme) FROM nouvelle_table)
```

Remarque : La condition `WHERE` utilise la table (nommée `"nouvelle_table"` à l'aide d'un `"AS"`).