BDD4 : Langage SQL

1 Introduction

Conjointement à la définition des bases de données relationnelles, un langage spécifique permettant à l'utilisateur d'effecteur des requêtes sur une base a été développé. Ce langage est nommé SQL pour Structured Query Language. Il existe de nombreux gestionnaires de base de données, différents par bien des aspects mais permettant tous de comprendre les requêtes écrites dans ce langage.

SQL a été standardisé par l'ANSI en 1986 puis par l'ISO en 1989, 1992, 1999, 2003 et 2008. Ces standards successifs ont étendu les capacités du langage. En plus de ces standards, SQL possède de nombreux dialectes spécifiques à chaque SGBD. En effet, le développement des SGBD s'est déroulé en parallèle, voire a précédé la standardisation de la norme SQL. En conséquence, seul un sous-ensemble de la norme est supporté par la grande majorité des SGBD (en particulier les SGBD open-source). Ce sous-ensemble correspond en fait à la norme SQL-92. C'est donc cette version de la norme SQL que nous allons présenter ici.

On peut distinguer dans SQL trois sous-langages particuliers :

- le langage de manipulation de données (DML pour Data Manipulation Language), qui permet d'obtenir des informations et de les mettre à jour ;
- le langage de description de données (DDL pour Data Description Language), qui permet de créer des relations, de modifier leur schéma etc.
- le langage de contrôle des données (DCL pour Data Control Language) qui permet de restreindre l'accès des données.

Lors des travaux pratiques, nous utiliserons une application fournissant une interface graphique pour effectuer nos requêtes : SQLite Admin.

Nous supposerons dans ce cours pour traiter des exemples que nous gérons une chaîne de locations de véhicules. Nous disposerons donc de deux tables Agence et Vehicule dont les schémas relationnels sont :

Agence : (**id_Agence**,$\mathbb{N}$), (ville,$\mathcal{S}$), (dep,$\mathbb{N}$)

Vehicule : (**id_Vehicule**,$\mathbb{N}$), (marque,$\mathcal{S}$) , (modele,$\mathcal{S}$), (couleur,$\mathcal{S}$), (km,$\mathbb{N}$), (neige,$\mathcal{B}$), (id_agence,$\mathbb{N}$)

2 Bloc de qualification

La structure de base est le bloc de qualification introduit par le mot-clé **SELECT**

```
SELECT Ai,..., An – colonnes et agrégations
FROM R – relation
WHERE F – assertion
GROUP BY A – regroupement
HAVING H – assertion
ORDER BY T – tri
; – finit la requête
```

Les clauses GROUP BY, ORDER BY et HAVING et WHERE sont optionnelles.

Nous pouvons détailler la sémantique des clauses FROM, WHERE et SELECT :

- SELECT désigne les attributs des n-uplets (ou enregistrements) qui doivent apparaître dans la solution *(Quelles colonnes je veux ?)*.
- FROM permet de préciser quelles sont la ou les relations sur lesquelles on pose la requête '*(Dans quelles tables ?)* ;
- WHERE permet de poser une assertion qui doit être vérifiée par les lignes de la table solution *(Quels critères ?)* ;

Les trois dernières clauses nous font sortir du calcul et de l'algèbre relationnelle :
- La clause GROUP BY définit comment regrouper des n-uplets (agrégation)
- La clause HAVING impose une condition sur les groupes (i.e. permet d'éliminer l'intégralité d'un groupe, en se basant sur la valeur d'un agrégat calculé sur ce groupe).
- La clause ORDER BY définit les critères de tris des résultats.

Le bloc de qualification simple (ne contenant que les clauses SELECT, FROM et WHERE) est donc une combinaison d'une projection et d'une sélection éventuellement appliquée à une jointure.

3 Opérations sur une seule table

3.1 Projection

Une projection simple aura en particulier comme syntaxe :

```
SELECT Ai, ..., Ap
FROM R;
```

Remarquons que l'on peut obtenir avec une projection des n-uplets identiques, en particulier si on ne sélectionne pas la clé primaire parmi les colonnes de la projection. Pour éviter cela, on peut utiliser le mot clé **DISTINCT** placé après SELECT.

Exercice 1 : Donner la requête SQL qui renvoie la liste des véhicules en affichant leur marque et leur modèle

3.2 Sélection

```
SELECT *
FROM R
WHERE F;
```

Ici, * désigne l'ensemble des colonnes de la relation R. F est une expression logique permettant de sélectionner les colonnes.

L'expression logique peut contenir des constantes et des noms de colonnes de la relation R. On la construit avec les opérateurs suivants :
- <, <=, =, >=, >, <> ;
- AND, OR, NOT ;
- IN, par exemple IN ('Peugeot', 'Renault') ;

Exercice 2 : Donner la requête SQL qui renvoie la liste des véhicules ayant plus de 30000 km.

Exercice 3 : Donner la requête SQL qui renvoie la liste des véhicules ayant entre 10000 et 50000 km, et dont la marque commence est Peugeot ou la couleur est verte. On affichera la marque, le modèle, le kilométrage et la couleur de chaque véhicule.

3.3 En-têtes de colonnes

L'entête des colonnes de la table résultat d'une requête est soit le nom de l'attribut correspondant, soit une expression de calcul contenant le nom d'un attribut.

Pour une meilleure présentation, on peut changer le nom des colonnes de la table résultat en utilisant le mot-clé **AS**

```
SELECT col1 AS 'Nouveau nom 1', col2 AS 'Nouveau nom 2'
FROM R
... ;
```

L'entête du résultat affiché portera les noms Nouveau nom 1 et Nouveau nom 2 au lieu de col1 et col2.

Exercice 4 : Donner la requête SQL qui renvoie la liste des véhicules dont le modèles est une Renault en affichant la marque, le modèle et le kilométrage avec des colonnes dont le nom est donné par la liste précédente.

3.4 Tris de données

Pour fournir un résultat trié suivant certaines colonnes, il existe une clause particulière, la clause **ORDER BY**.

```
ORDER BY liste des attributs
```

où chaque élément de la liste est un attribut suivi de ASC (ordre ascendant) ou DESC (ordre descendant). Par défaut, c'est ASC qui est utilisé.

Exercice 5 : Donner la requête SQL pour demander quel la liste des véhicules triés par kilométrage décroissant puis par marque croissante (dans l'ordre alphabétique) :

4 Opérations sur plusieurs tables

4.1 Sélection sur plusieurs tables

```
SELECT A1,..., Ap
FROM R1, R2,..., Rk
WHERE F;
```

Si l'on considère t1, . . . , tk des nuplet-s respectifs de R1, . . . , Rk, le n-uplet (t1,..., tk) sera qualifié s'il satisfait la formule logique F.

La table résultat sera obtenue en effectuant une projection de (t1, . . . , tk) sur A1, . . . , Ap. La requête est donc semblable à $\pi_{A_1,...,A_p}(\sigma_F(R_1 \times ... \times R_k))$

Dans le cas d'attributs provenant de tables différentes et portant le même nom, on les distingue en les préfixant par le nom de la relation.

Exercice 6 : Donner la requête SQL pour donner les marques et modèles des véhicules disponibles à l'agence d'Arles

4.2 Jointures

La seule jointure au programme est la jointure symétrique. Il s'agit de créer une jointure selon des critères d'égalité. On peut donc voir la jointure comme la composition d'un produit cartésien et d'une sélection. On utilise la syntaxe **JOIN...ON...** :

```
R JOIN S ON F
```

R JOIN S ON F établit la jointure de R et S sur le critère F (qui est une expression logique).
Le résultat de cette expression est évidemment une relation. On peut donc l'utiliser dans une sélection.

Exercice 7 : Donner la requête SQL pour donner les modèles des véhicules disponibles à l'agence d'Arles.

Exercice 8 : Donner la requête SQL pour donner la liste des véhicules dont le kilométrage est inférieur ou égal à 30 000 km et qui sont disponibles dans une agence du département 13.

5 Fonctions d'agrégation

Les opérateurs d'agrégation permettent d'effectuer des opérations sur certaines colonnes d'une relation. On retrouve dans SQL les mêmes opérateurs qu'en algèbre relationnelle étendue :

– SUM qui permet de sommer une colonne contenant des valeurs numériques
– AVG qui permet de moyenner une colonne contenant des valeurs numériques
– MIN et MAX qui retourne respectivement la plus petite et la plus grande valeur d'une colonne
– COUNT retourne le nombre de valeurs d'une colonne.

Si l'on note OP1, OP2, . . . , OPk des opérateurs d'agrégations, la syntaxe de leur utilisation est la suivante :

```
SELECT OP1(Ai),...,OPk(Aj)
FROM R
WHERE F;
```

Seul l'opérateur COUNT peut s'utiliser avec * : COUNT(*) permet de compter tous les n-uplets résultant d'une requête.

Exercice 9 : Donner la requête SQL pour donner le kilométrage moyen de tous les véhicules.

*Exercice 10 :*Combien de véhicules sont disponibles à l'agence de Marseille ?

6 Groupement de données et sélection par agrégation

Lorsqu'on souhaite utiliser une fonction d'agrégation dans une sélection, le seul bloc WHERE est insuffisant. On utilise alors le bloc GROUP BY ... HAVING ... :

```
SELECT colonne1, OP(colonne2)
FROM R
GROUP BY colonne1
HAVING fonction(colonne2) operateur valeur
```

Cela permet donc de ne garder que les colonnes de la table R en groupant les lignes qui ont des valeurs identiques sur la colonne « colonne1 » et que la condition de HAVING soit respectée.

Exercice 11 : Donner la requête SQL qui affiche le nombre de véhicules de chaque marque, en ne gardant que les marques pour lesquelles on possède plus de 5 véhicules.

Exercice 12 : Donner la requête SQL pour afficher la liste des ville des agences (on suppose que chaque ville ne contient qu'une seule agence) avec le nombre de véhicules disponibles dans chaque agence, en ne gardant que les agences qui disposent d'au moins 6 véhicules. Les résultats seront triés par ordre décroissant du nombre de véhicules total.

7 Sous-requêtes

Le résultat d'une requête est une table. On peut donc mettre le résultat d'une requête après un FROM, ou utiliser le résultat d'une fonction d'agrégation obtenue par une requête dans une clause WHERE. Cette technique, très puissante, s'appelle une sous-requête.

*Exercice 13 :*Donner la requête SQL affichant la marque et le modèle du véhicule noir ayant le plus grand kilométrage.

*Exercice 14 :*Donner la requête SQL affichant le véhicule ayant le deuxième kilométrage le plus élevé du parc automobile.